VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Bakalárska práca

2012 Peter Vrba

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizace grafových struktur
Graphs Visualization

2012

Peter Vrba

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Peter Vrba**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizace grafových struktur
Graphs Visualization

Zásady pro vypracování:

Cílem práce je implementace vizualizačního nástroje, který bude sloužit k zobrazení grafové struktury nad vybranou kolekcí. Vybranou technologií bude Microsoft .NET, Silverlight.

1. Přehled existujících nástrojů a technik pro vizualizaci informace, s bližším zaměřením na vizualizaci grafu a podobnost dokumentů.
2. Návrh aplikace ve vybrané technologii.
3. Implementace a vyhodnocení výkonostích testů.
4. Zhodnocení dosažených cílů a provedených experimentů.

Seznam doporučené odborné literatury:

Matthew MacDonald: Pro Silverlight 4 in C#, Apress; 3 edition (November 9, 2010) ISBN13: 978-1-4302-2979-7

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Gajdoš, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

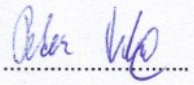
prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne.

Uviedol som všetky literárne pramene a publikácie z ktorých som čerpal.

V Ostrave dňa 02. 05. 2012



Podpis

Poďakovanie

Rád by som poďakoval môjmu vedúcemu bakalárskej práce Ing. Petrovi Gajdošovi, Ph.D za účinnú pedagogickú a odbornú pomoc a všetky ďalšie odborné rady pri spracovaní tejto bakalárskej práce.

Abstrakt

Vizualizácia informácií je samostatná disciplína vedeckých štúdií, ktorá rieši problém vizuálnej reprezentácie kolekcií spravidla nenumerických veličín. Jej hlavným cieľom je sprostredkovať informáciu používateľovi jasne a efektívne cez grafické rozhranie. Kolekcie dát sú najčastejšie reprezentované rôznymi typmi grafov, ktoré môžu znázorňovať napríklad aj vzťahy medzi dokumentmi kolekcie.

Táto bakalárska práca poskytuje teoretický úvod do problematiky hľadania podobností v rôznych typoch médií. Jej praktická časť sa zameriava na návrh a implementáciu komponenty, ktorá by bola schopná vizualizovať závislosti medzi dokumentmi za účelom odhalenia plagiátorstva. Táto komponenta je postavená na technológiách .NET a Silverlight.

Kľúčové slová

Hľadanie podobností, Silverlight, vizualizácia, .NET, graf, kolekcie

Abstract

Information visualization is a separate discipline of scientific studies that addresses the problem of visual representation of collections generally non-numeric values. Its main purpose is to convey information to the user clearly and effectively through graphical interface. Collections of data are usually represented by different types of graphs that can show for example a relationship between documents of collection.

This bachelor thesis provides a theoretical introduction to problems of search for similarities in the different media types. The practical part is focused on the design and implementation of component that would be able to visualize the dependencies between documents in order to detect plagiarism. This component is based on technology .NET and Silverlight.

Key words

Similarity detection, Silverlight, visualization, .NET, graph, collection

Zoznam použitých skratiek

AJAX	Asynchronous JavaScript and XML
FLV	Flash Video
GNU	Gnu is Not Unix
GUI	Graphical User Interface
HTML	HyperText Markup Language
RIA	Rich Internet Application
SDK	Software Development Kit
SVD	Singular Value Decomposition
SWF	ShockWave Flash
WMA	Windows Media Audio
WMV	Windows Media Video
WPF	Windows Presentation Foundation
WYSIWYG	What You See Is What You Get
XAML	Extensible Application Markup Language
XAP	XAML Application Package
XML	Extensible Markup Language
ZIP	Zipped archive

Obsah

1. Úvod

Žijeme v informatickom veku, kde majú dáta veľkú hodnotu, dokonca až takú, že sa s nimi obchoduje. Existujú firmy, ktoré sa špecializujú len na zber a spracovanie informácií. Taktiež sú firmy, ktoré veľmi ochotne za tieto informácie zaplatia. Samotné dáta by boli bezcenné pokiaľ by sa nedali analyzovať. Dnes sa analyzuje napríklad podobnosť dokumentov, hry na mobilné telefóny, regionálne rozloženie návštevníkov portálu, aktivita týchto návštevníkov alebo vyťaženie serverov. Výsledky týchto analýz je potreba vizualizovať a k tomu slúži veľké množstvo nástrojov. Vo väčšine prípadov sú buď špecificky zamerané alebo veľmi jednoduché a neobsahujú pokročilé funkcie. Len veľmi málo komponent je dostatočne flexibilných a zároveň poskytujú určitú úroveň funkcionality.

Táto bakalárska práca sa zaoberá problematikou analyzovania dát a ich vizualizáciou. Poskytuje zbežný pohľad na problematiku plagiátorstva a hľadanie zhodnosti dokumentov. Taktiež poskytuje prehľad technológií vhodných na vizualizáciu dát. Jej hlavným cieľom je navrhnúť grafickú komponentu, ktorá dokáže vizualizovať kolekciu dokumentov a znázorniť vzťahy medzi jednotlivými dokumentmi, ako napríklad podobnosť. Táto komponenta by mala okrem vizualizácie poskytovať aj možnosť ďalej s touto kolekciou pracovať, analyzovať ju, vyhľadávať v nej a mapovať jej vývoj. Návrh tejto komponenty by mal byť flexibilný a dať sa použiť na ľubovoľný typ kolekcie. Komponenta nebude určená na prácu s veľkými kolekciami dát. Zamýšľaný počet je rádovo v tisíckach, ideálne do 3000, pretože táto komponenta nie je určená na hľadanie podobností, ale len na vizualizáciu výsledkov. Väčší počet dokumentov nemá z hľadiska prehľadnosti zmysel vizualizovať. Zamýšľané použitie je v nástroji na hľadanie podobností, ktorý by dostal skúmaný dokument a našiel by všetky jemu podobné dokumenty. Tento výstup by vrátil v podobe xml súboru, ktorý spracuje a zobrazí naša komponenta.

Výsledkom tejto bakalárskej práce bude implementácia navrhnutej komponenty s použitím technológií Microsoft .NET a Silverlight. Táto implementácia bude podrobená výkonnostným testom, ktoré budú v závere vyhodnotené.

2. Teoretická časť

Táto kapitola je venovaná problematike plagiátorstva a metódam hľadania podobností rôznych médií. Taktiež tu poskytujeme prehľad vhodných technológií na vizualizáciu dát. Zamerali sme sa na dve najvhodnejšie technológie: Adobe Flash a Microsoft Silverlight, ktoré popíšeme a vzájomne porovnáme. Keďže je naša komponenta vyvíjaná práve v technológií Silverlight, tak sa jej budeme venovať viac dopodrobna.

Cieľom tejto práce nie je hľadanie podobností dokumentov, ale tvorba komponenty, ktorá bude schopná vizualizovať tieto podobnosti. Z tohto dôvodu nebudeme rozoberať metódy hľadania podobností dopodrobna, ale iba zbežne a poskytneme referencie na zdroje, kde je možnosť si bližšie túto problematiku naštudovať.

2.1 Metódy hľadania podobností

Hľadanie podobnosti dokumentov má využitie hlavne v odhaľovaní plagiátorstva. Pri dostatočne veľkej množine zanalyzovaných dokumentov sa dá dohľadať relatívne presne, odkiaľ autor čerpal. Na druhú stranu problematika plagiátorstva je komplikovaná, pretože je ťažké stanoviť hranicu plagiátorstva. Ak autori píšu o rovnakej téme, je samozrejmé, že ich práce budú používať rovnaké výrazy, popisovať tie isté javy a fakty bez ohľadu na to, či sa jedná o plagiátorstvo alebo nie. Rozdiel je len v miere zhodnosti textov ako takých, pričom ak je plagiátor šikovný a dá si záležať, dokáže text zmeniť tak, že oklame aj algoritmy na hľadanie podobností, pretože sa bude jednať o odlišný text ako originál. Samotné algoritmy na hľadanie podobnosti nie je možné v reálnej praxi použiť, musia byť vylepšené tak, aby nebrali do úvahy určité časti textov ako sú napríklad úvodné strany, hlavičky a podobne. Je zrejmé, že tieto môžu byť a veľakrát aj budú identické, preto je dôležité ich ignorovať, aby nám umelo nezvyšovali mieru zhodnosti dokumentov.

Podobnosť sa dá určiť aj pri iných typoch dát ako len čistý text. Bežné je porovnávanie podobností napríklad obrázkov, ktoré sa tiež uplatňuje v oblasti plagiátorstva, pretože autor mohol skopírovať aj obrázky. Spoločnosť Google využíva tieto metódy pri vyhľadávaní obrázkov. Užívateľ môže jednoducho nahrať svoj obrázok a vyhľadávač Google Images[1] sa mu pokúsi nájsť podobné obrázky na internete. Zhodnosť dvoch zvukových záznamov je tiež merateľná a dokonca sa dnes veľmi často používa. Asi najlepším príkladom je služba Shazam[2], ktorá sa preslávila na mobilných telefónoch. Funguje takým spôsobom, že si cez mikrofón na mobilnom telefóne nahrá pár sekundový záznam, ten pošle na svoj server a porovná ho so svojou databázou pesničiek. Ak nájde zhodu, tak vráti informácie o danej skladbe.

V tejto podkapitole stručne popíšeme postupy pri určovaní podobností rôznych médií a uvedieme referencie na zdroje, kde je možné túto problematiku bližšie naštudovať.

[1] Google Images – vyhľadávač obrázkov. Požadovaný obrázok je možno špecifikovať využitím kľúčových slov, ktoré sa s nim spájajú, alebo pomocou podobného obrázku. (web: images.google.com)
[2] Shazam – služba na identifikáciu pesničiek. (web: www.shazam.com)

2.1.1 Textové dokumenty

Metód na hľadanie podobností textových dokumentov je viacero. Takmer všetky majú základné črty rovnaké, ako napríklad predspracovanie dokumentov do jednotného formátu, čím dochádza k ich zanalyzovaniu a odstráneniu nepotrebných častí. Základnou rozlišovacou jednotkou dokumentu môže byť jeden znak, slovo, alebo term[3]. Samotné dokumenty sú potom najčastejšie reprezentované pomocou m-dimenzionálnych vektorov, ktoré znázorňujú výskyt jednotlivých slov/termov v dokumente. Nakoniec vznikne kolekcia n-dokumentov, z ktorých je každý reprezentovaný ako m-rozmerný vektor. Vo výsledku máme teda maticu N x M, ktorá sa nazýva matica termov. Tá však môže mať veľké rozmery vzhľadom na počet termov v jednom dokumente a veľkosť kolekcie dokumentov, takže nie je možné v praxi takúto maticu použiť priamo na porovnávanie zhodnosti dokumentov. Tento problém sa rieši zredukovaním dimenzií vektorov pomocou SVD rozkladu.

Jednotlivé termy sú spracované váhovou metódou, ktorá pridelí určitú váhu každému termu. Váhové metódy sa líšia v komplexnosti, kde najjednoduchšie označujú termy len binárne podľa toho či sa v danom dokumente vyskytujú alebo nie, zatiaľ čo zložitejšie metódy berú do úvahy aj počet výskytov daného termu v celej kolekcií dokumentov [1]. Existujú aj metódy, ktoré sú nezávislé na konkrétnom jazyku a dokážu nájsť zhodu aj medzi dvoma dokumentmi, ktoré sú napísané v rozdielnych jazykoch. Dosahujú toho pomocou viacjazyčného slovníka Thesaurus EUROVOC [2].

2.1.2 Obrázky

Pri hľadaní podobnosti dvoch obrázkov sa tiež používa predspracovanie, najčastejšie formou odtlačkov (fingerprint). Keďže obrázok je v podstate len matica pixelov, tak by bolo veľmi časovo náročné porovnávať kompletne tieto matice. Okrem zníženia náročnosti na výpočet majú odtlačky ešte jednu veľkú výhodu a tou je zjednodušenie obrázku, čím zaniknú nepodstatné detaily, ktoré by v prípade porovnávania po pixeloch mali význam. Tieto nepodstatné detaily môžu byť tiež: iný rozmer toho istého obrázku alebo jeho pootočenie, výrez, iný odtieň, a podobne.

Niektoré metódy tvorby odtlačkov prevedú obrázky do monochromatických farieb napríklad odtieňov šedej, iné pracujú s farebnými obrázkami. Odtlačok môže vzniknúť použitím Gaussovho mixu, kde sa dva obrázky porovnávajú pomocou odchýlky ich mixov [3]. Inou metódou je napríklad zmenšenie obrázku na malé rozmery, napríklad 4x4 pixelov, čo sa dá docieliť jedine spriemerovaním farebných odtieňov. Takýto odtlačok sa potom otočí tak, aby najsvetlejšia strana bola vždy na hornej strane. Podobná metóda je aj rozdelenie obrázku na určitý počet segmentov, kde sa farba každého segmentu spriemeruje oddelene. Táto metóda je veľmi účinná pri obrázkoch, ktoré majú iné rozmery a poradí si aj s roztiahnutými obrázkami [1].

[3] Term – slovné spojenie alebo fráza. Pozostáva z viacerých slov a zoznam termov v dokumente sa získava pomocou tzv. term-extračných metód, kedy sa vyextrahujú najčastejšie používané slovné spojenia

2.1.3 Zvukové záznamy

Hľadanie zhodných zvukových záznamov (ako príklad budem uvádzať pesničky) je tiež založené na odtlačkoch v databáze piesni. Každá pesnička musí byť predspracovaná. Najskôr sa vytvorí spektogram[4] reprezentujúci celú pesničku, ktorý má na horizontálnej osi čas a na vertikálnej osi frekvenciu. Jeho tretím rozmerom je intenzita danej frekvencie v konkrétnom čase a je znázornená farbou. Jeden takýto spektogram uchováva veľké množstvo informácií a väčšina z nich sú nepotrebné, takže je výrazne zredukovaný iba na tie najintenzívnejšie frekvencie. Tým vznikne dvojrozmerný graf, pretože sa odfiltrovali všetky nepotrebné frekvencie (s malou intenzitou) a tak sa intenzita stáva nepodstatnou a záleží len na konkrétnej frekvencií v konkrétnom čase. Z takéhoto grafu sa vyberú určité frekvencie, z ktorých sa vytvorí hash[5] a tieto frekvencie sa nazývajú „kotvové frekvencie“. K danej frekvencií sa nájdu okolité frekvencie, ktoré nasledujú za ňou na časovej osi. Tie sa potom spriemerujú vo vzťahu ku kotvovej frekvencií, kde sa zohľadňuje: frekvencia kotvy, frekvencia okolitého bodu, časový bod kotvy a časový rozdiel medzi okolitým bodom a kotvou. Každá pesnička (odtlačok) obsahuje veľké množstvo týchto hashov. Zvukový záznam, ktorý sa ide porovnávať s databázou piesni musí tiež prejsť týmto procesom a vznikne jeho odtlačok, ktorý býva spravidla menší, pretože stačí 10 sekúnd na to, aby bolo možné presne identifikovať pesničku, z ktorej daný záznam pochádza [4].

2.2 Prehľad technológií

V tejto podkapitole sa budem zaoberať technológiami vhodnými na vizualizáciu dát. Pre náš účel sú vhodné RIA technológie. V nasledujúcich podkapitolách si vysvetlíme, čo znamená pojem RIA technológia a popíšeme dve najpoužívanejšie: Adobe Flash a MS Silverlight. Keďže je naša komponenta vyvíjaná v technológií Silverlight, budeme sa jej venovať detailnejšie.

2.2.1 RIA technológie

Tento pojem označuje takzvané Rich Internet Application, čo znamená internetová aplikácia s bohatým obsahom. V internetovej terminológií rozlišujeme základné dva typy klientov: tučný klient a tenký klient. Tučný klient je aplikácia, ktorá celá prebieha na strane klienta, čo znamená, že spracovanie dát aj vykresľovanie užívateľského rozhrania prebieha u klienta. Internet slúži v podstate len na stiahnutie dát, s ktorými bude tento klient pracovať, pričom spravidla už potom internet nie je využívaný. Na druhú stranu, tenký klient slúži na prezentovanie dát spracovaných na strane servera. Poskytuje užívateľské rozhranie (zvyčajne jednoduché) a každá požiadavka na spracovanie dát je poslaná na server, ktorý ju vykoná a vráti výsledok. RIA aplikácia je umiestnená niekde medzi týmito dvoma klientmi. Podľa definície je to webová aplikácia, ktorej užívateľské rozhranie je bohaté na funkcie a je spracované na strane klienta, zatiaľ čo biznis logika je zabezpečená prostredníctvom serverových služieb (services) [5]. Hlavným rysom je práve bohaté užívateľské rozhranie, ktoré zvyčajne poskytuje všetok komfort a možnosti, na aké je užívateľ zvyknutý z desktopových aplikácií. Takéto webové aplikácie posielajú dáta na pozadí,

[4] Spektogram – trojrozmerný graf používaný na zobrazenie zmien nejakej veličiny v čase
[5] Hash – výstup „hashovacej“ funkcie, ktorá z reťazca vstupných dát vytvorí krátky výstupný reťazec

takže sa nemusia po každej požiadavke znovu načítať ako bežné stránky. Okrem toho môžu využívať hardware klienta a uchovávať určité množstvo dát lokálne.

RIA technológie zabezpečujú spustenie RIA aplikácií. V súčasnosti sú najpoužívanejšie tieto technológie: Adobe Flash, Microsoft Silverlight a rozličné JavaScript Frameworky + AJAX. Zvyčajne vyžadujú nainštalovanie binárnych pluginov[6] do webového prehliadača (Flash, Silverlight), ale nie je to podmienkou (AJAX+JavaScript).

2.2.2 Adobe Flash

Adobe Flash vznikol v deväťdesiatych rokoch ako webová technológia, ktorá dokáže vykonávať činnosti, aké nedokáže HTML s JavaScriptom, napríklad animované filmy a hry. Veľmi rýchlo sa začala táto technológia používať na tvorbu prezentácií, hier a hlavne internetových reklám. Na svoj beh v internetovom prehliadači potrebuje nainštalovaný plugin s názvom Adobe Flash Player. Ak je na strane klienta nainštalovaný tento plugin, tak je pridanie Flashu na webové stránky veľmi jednoduché a v princípe sa neodlišuje od pridania obrázku. Samozrejme kód pre vloženie je viac komplexný ako v prípade obrázku [6].

V súčasnosti je Flash na internete veľmi rozšírený vďaka hernej scéne, kde má prakticky monopol. Na druhú stranu, jeho podiel na internetovej reklame klesá zásluhou HTML5, ktorý je podporovaný „veľkými hráčmi“ ako sú Apple, Google a Microsoft. Stále je však významným hráčom na poli internetového videa, kde spomedzi konkurenčných technológií poskytuje najväčšie možnosti a najširšie spektrum „kompatibilných“ užívateľov. Táto technológia umožňuje zachytávať vstup z myši, klávesnice, webovej kamery a mikrofónu. Taktiež vie zobrazovať rastrovú aj vektorovú grafiku v takmer všetkých formátoch, či streamovať video a zvuk prostredníctvom vlastného formátu FLV. Flash je schopný spracovať video vo viacerých formátoch a aktuálne najpoužívanejším formátom je H.264. Logika aplikácie je napísaná v programovacom jazyku ActionScript, ktorý je objektovo orientovaný. Práve ten prispel k jeho obľube na poli internetovej reklamy a hier, pretože poskytuje široké možnosti, interakciu a v kombinácií s vektorovou grafikou aj nízku veľkosť aplikácie. Flash aplikácie sú distribuované v komprimovanom formáte SWF, ktorý obsahuje všetky obrázky, text aj animácie. Vďaka tomu majú nižšiu veľkosť a sú ťažšie indexovateľné pre vyhľadávače. Na definovanie animácie sa používa framerate[7], kde je dĺžka trvania nejakého úseku vyjadrená počtom snímkov, čo spôsobuje problémy pri rozličných výkonoch a vyťaženiach počítačov. V takom prípade musí oneskorenie kompenzovať programátor. Flash aplikácie dokážu fungovať na veľkom množstve webových prehliadačov a operačných systémov, vrátane Mac OS X a Linux [7].

2.2.3 Microsoft Silverlight

Predtým než budeme popisovať technológiu Silverlight, je potrebné vysvetliť pojem WPF, z ktorého Silverlight vychádza. WPF je priamym nástupcom WinForms, ktoré Microsoft používal vo svojich Windows až do verzie XP (tie boli schopné používať aj WPF po doinštalovaní Service Pack 2) a slúžia na tvorbu užívateľského rozhrania. Ďalšia verzia Windows s názvom Vista už

[6] Plugin – zásuvný modul, ktorý je v binárnej podobe a rozširuje funkcionalitu aplikácie, ktorá ho importuje
[7] Framerate – počet snímkov za sekundu

mala integrovanú podporu WPF. Cieľom bolo umožniť tvorbu užívateľského rozhrania, ktoré by bolo ľahko upravovateľné, umožňovalo by tvorbu vlastných ovládacích prvkov, malo by podporu hardvérovej akcelerácie, vedelo by zobrazovať 2D/3D grafiku, rastrové a vektorové obrázky. Vo WPF je dizajn oddelený od aplikačnej logiky vďaka značkovaciemu jazyku XAML.

Silverlight vznikol ako verzia WPF, ktorá by bola schopná fungovať vo webovom prehliadači. Pred oficiálnym uvedením bol označovaný ako WPF/E, presnejšie Windows Presentation Foundation Everywhere. Microsoft zamýšľa Silverlight ako technológiu pre koncových užívateľov, ale tiež ako technológiu pre biznisové riešenia. Už od začiatku bolo zrejmé, že Silverlight bude cielený hlavne na biznisové riešenia, pretože má integrované funkcie určené pre mechanizmus doručovania biznisových služieb [8].

Na poli hier a prehrávačov videa sa Silverlight veľmi nepresadil, ale na druhú stranu má väčšinové zastúpenie v oblasti biznis aplikácií, na ktoré je lepšie pripravený než Flash. Na jeho programovanie je možné použiť celú radu programovacích jazykov, ako napríklad Visual C#.NET a Visual Basic.NET, pričom umožňuje aj skriptovanie na strane klienta pomocou JavaScriptu. C# a Visual Basic je možné použiť na napísanie manažovaného kódu, ktorý beží na platforme Microsoft .NET Framework a využíva všetky jeho možnosti, hlavne technológie určené na prácu s dátami a komunikáciu so serverom.

Na tvorbu animácií slúži tzv. WPF Animation Model, ktorý je časovo založený (na rozdiel od snímkovo orientovaného Flashu), takže stačí nastaviť počiatočné a koncové podmienky, časový úsek a WPF sa už sám postará o animovanie. Programátor nemusí prepočítavať jednotlivé snímky, ale radšej pracuje s animáciou intuitívnejším spôsobom. S formátmi videa je na tom slabšie ako Flash. Ich počet je výrazne nižší (hlavnými formátmi sú WMV a WMA), ale na druhú stranu vďaka podielu operačného systému Windows na trhu sú dosť rozšírené a Microsoft poskytuje zdarma aj SDK Encoder pre produkovanie týchto formátov, takže nie je problém s ich kódovaním, ani s vytvorením vlastného kódovania.

Aplikácia je po skompilovaní uložená do ZIP archívu, ktorý má koncovku XAP. Silverlight ponúka dva spôsoby ako začleniť aplikáciu do webovej stránky. Prvý je pomocou HTML tagu „object“ a druhý spôsob je pomocou Silverlight.js, ktorý je súčasťou SDK. Obidva spôsoby vloženia povedú k rovnakému výsledku a tým je vytvorenie elementu „object“. Silverlight má pluginy pre všetky významné webové prehliadače a operačné systémy (implementácia pre operačný systém GNU/Linux má názov Mono a Microsoft na nej spolupracuje s firmou Novell). Silverlight dokážu spracovať aj mobilné telefóny s operačným systémom Windows Phone 7.

Tvorba Silverlight aplikácií prebieha hlavne za použitia dvoch vývojových nástrojov. Prvým je Visual Studio (aktuálne vo verzií 2010), ktorý slúži ako vývojové prostredie pre celý .NET Framework a používa sa hlavne na tvorbu logickej vrstvy aplikácie. Zvláda aj tvorbu prezentačnej vrstvy, či už prostredníctvom WYSIWYG[8] editoru alebo ručne pomocou úpravy XAML kódu. Širšie možnosti úpravy dizajnu však ponúka až druhý vývojový nástroj. Expression Blend (aktuálne vo verzií 4) je nástroj pre dizajnérov, v ktorom je možné pohodlne upravovať a tvoriť

[8] WYSIWYG – What You See Is What You Get – „Čo vidíš, to dostaneš“ je označenie pre editor v ktorom je výsledný dizajn zobrazovaný presne tak ako ho uvidí užívateľ

prezentačnú vrstvu Silverlight aplikácie. Umožňuje aj tvorbu desktopových rozhraní založených na technológií WPF. Opäť umožňuje tvorbu pomocou WYSIWYG editoru alebo ručnou úpravou XAML kódu.

XAML je značkovací jazyk určený na popis užívateľského rozhrania WPF aplikácií. Vychádza z jazyka XML, ktorého syntax a sémantiku prísne dodržiava. Podobne ako XML je ľahko čitateľný pre ľudí aj pre počítač. Definuje sa ním prezentačná vrstva vo vektorovom formáte, v ktorej je logika oddelená od dizajnu. Užívateľské rozhranie sa skladá z dvoch súborov. Prvým je súbor s koncovkou XAML, v ktorom je uložená deklarácia rozhrania a druhý súbor má koncovku CS (v prípade programovacieho jazyka C#, koncovka sa môže meniť v závislosti na použitom programovacom jazyku) a je v ňom uložená logická časť rozhrania [9]. Jednotlivé triedy WPF sú v XAML reprezentované elementmi s rovnakým názvom. Na príklade demonštrujem rozdiel vo vytváraní ovládacieho prvku s názvom Slider dynamicky v jazyku C# a v jazyku XAML:

C#:

```
1.  Slider slider = new Slider();
2.  slider.Maximum = 100;
3.  slider.Minimum = 1;
4.  slider.Value = 50;
5.  slider.Background = new SolidColorBrush(Colors.White);
6.  this.MainCanvas.Children.Add(slider);
```

XAML:

```
1.  <Slider x:Name="slider" Maximum="100" Minimum="1" Value="50" Background="White"
    />
```

Ďalšou veľmi užitočnou funkciou Silverlightu je DataBinding. Ten poskytuje jednoduchý spôsob ako prezentovať a spravovať dáta v klientskej aplikácií. Zvláda proces presunutia dát od zdroja k cieľovému prvku užívateľského rozhrania. DataBinding dokáže pracovať v jednosmernom režime (dáta idú len od zdroja k cieľovému prvku rozhrania) alebo v obojsmernom režime, kedy môže ovládací prvok ovplyvňovať dáta na zdroji. Je možné naprogramovať aj konvertor, ktorým sa dáta zo zdroja automaticky upravia predtým než budú poslané do prezentačnej vrstvy, čo je užitočné napríklad pri konvertovaní meny na aktuálnu menu klienta alebo pri dosadzovaní názvov namiesto hodnôt v číselníku [10].

3. Návrh riešenia

V tejto kapitole je rozpísaný celý návrh riešenia od požiadaviek na funkčnosť, cez návrh architektúry až po navrhnutie grafického užívateľského rozhrania.

3.1 Špecifikácia funkcionality

V tejto podkapitole popíšeme požiadavky, ktoré by mala naša komponenta spĺňať. Základnou požiadavkou je, aby bola interaktívna. Jednotlivé uzly by sa mali dať selektovať a graf by sa mal dať jednoduchým pohybom myši upravovať a preorganizovať. Predpokladá sa spracovanie väčšieho počtu uzlov, takže je na mieste požiadavka na skrývanie uzlov, pomocou ktorej by bolo možné skryť nepotrebnú časť grafu a neskôr ju znova zviditeľniť.

Podobne funguje aj funkcia na odstránenie z kolekcie, ktorá na rozdiel od skrytia úplne odstráni selektované uzly z kolekcie. Hlavné využitie to má pri veľkých kolekciách, ktoré sú pre svoj veľký počet uzlov rozložené tesne vedľa seba. Takéto rozloženie je veľmi neprehľadné a ak chce užívateľ preskúmať určitú časť, tak jednoducho odstráni všetko nepotrebné z kolekcie a dá si znova rozmiestniť kolekciu, takže táto zmenšená kolekcia bude rozložená oveľa prehľadnejšie. Tieto dve funkcie sa budú dať samozrejme kombinovať a užívateľ bude schopný skrývať a znova zobrazovať uzly z tejto zmenšenej kolekcie. Samozrejmosťou bude aj funkcia reset, ktorá navráti všetky uzly späť do kolekcie.

Ďalej by mala mať komponenta možnosť inverznej selekcie, čo bude veľmi praktické, ak chce užívateľ pracovať len so zmenšenou kolekciou. Bude mu stačiť vybrať to, s čím chce pracovať, vykonať inverzný výber a odstrániť vybrané uzly. Taktiež by malo byť možné nastaviť zobrazovanie hrán od určitej úrovne, čím by sa dali prehľadne vyfiltrovať hrany podľa určitej sily spojenia (váha hrany). Meniteľná by mala byť aj mierka veľkosti zobrazovaných uzlov, čo dovoľuje užívateľovi sprehľadniť si graf.

3.1.1 SubGraphs

Subgraph alebo v preklade podgraf, je podmnožina vrcholov grafu. V našom prípade to budú také vrcholy, ktoré sú vzájomne spojené hranou tak, aby platilo, že neexistuje vrchol z podgrafu A, ktorý by bol hranou spojený s iným vrcholom v podgrafe B. Každý podgraf bude odizolovaný od ostatných podgrafov, pretože ak by medzi nimi bola hrana, jednalo by sa o jeden podgraf.

Komponenta bude schopná pri načítavaní dát automaticky roztriediť jednotlivé uzly do takýchto podgrafov. Na bočnej lište bude zobrazený zoznam všetkých podgrafov spolu s informáciou o počte uzlov v každom z nich. Následne bude možné jednoducho selektovať celé podgrafy a pracovať s nimi.

Táto funkcionalita bude dostupná z už spomínaného zoznamu, ale aj prostredníctvom kontextovej ponuky pod pravým tlačítkom myši, kde užívateľ vyberie konkrétny uzol, klikne naň pravým tlačítkom a dá selektovať všetky uzly, čo sú s ním spojené.

3.1.2 Search bar

Súčasťou komponenty by mal byť aj vyhľadávací riadok s podporou tagov. Mal by slúžiť na pokročilé selektovanie uzlov, kde si užívateľ môže zadať vlastné kritéria výberu a komponenta následne vyberie všetky uzly, ktoré spĺňajú dané kritéria. Podpora tagov slúži na špecifikovanie kritérií. V prípade kníh, bude možné prehľadávať nie len podľa názvu knihy, ale aj podľa ceny, roku vydania, autora, počtu strán, kategórie a podobne. Prakticky podľa akéhokoľvek údaju, ktorý bude dostupný.

Číselné parametre sa budú dať špecifikovať presne alebo pomocou intervalov. Bude tak možné vyhľadávať napríklad knihy, ktoré sú novšie ako rok 2004. Všetky tieto kritéria sa budú dať vzájomne kombinovať, takže výsledný dotaz môže v prípade kníh vyzerať napríklad takto: knihy, ktoré obsahujú v názve slovo „apple“, s cenou do 5 eur, vydané v rokoch 2002 až 2010, patriace do kategórie „Technika“ a majú počet strán väčší ako 200.

Vyhľadávanie sa bude zadávať formou Query Expression[9] a programátor si bude môcť nastaviť ľubovoľné názvy tagov. Odporúčané sú krátke označenia v anglickom jazyku. Výraz pre vyššie opísané kritériá by mohol vyzerať nasledovne:

name:”apple”; price<: 5.0; date-from: 2002-01-01; date-to: 2010-12-31; category: “Technika”; pages>: 200;

Jeden tag sa bude skladať z dvoch častí oddelených dvojbodkou, kde prvá časť bude názov kritéria a druhá bude hodnota. Jednotlivé tagy budú od seba oddelené bodkočiarkou. Odporúčaný zápis číselných kritérií je pomocou znakov „<“ a „>“. V prípade ceny tak budú možné tieto tri zápisy:

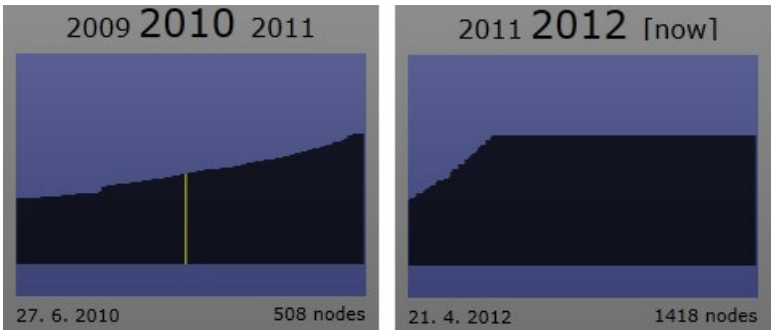
1. **price: 5.0** = presne 5 eur,
2. **price<: 5.0** = menej ako 5 eur
3. **price>: 5.0** = viac ako 5 eur

3.1.3 Progress graph

Požiadavkou na komponentu je funkcia zobraziť vývoj načítanej kolekcie s možnosťou vracania sa v čase. Progress graph (vývojový graf) je funkcia slúžiaca na zmapovanie vývoja danej kolekcie. Už počas načítania kolekcie z xml prebehne jej chronologické zoradenie. Každý rok bude reprezentovaný jedným poľom. Všetky načítané uzly budú roztriedené do týchto polí a potom zoradené podľa dátumu.

Vizuálna reprezentácia je formou grafu, ktorého vodorovná os znázorňuje dátum a zvislá os znázorňuje počet uzlov. Pod grafom bude zobrazený počet všetkých uzlov. Zobrazovať sa bude vždy len rozpätie jedného roku a medzi rokmi sa bude dať veľmi jednoducho prepínať. Toto rozdelenie uľahčí ovládanie a bude prehľadnejšie. Užívateľ takto dostane prehľad o raste danej kolekcie. Tiež bude môcť kliknutím vybrať konkrétny dátum a počet uzlov sa zmení vzhľadom k vybranému dátumu.

[9] Query Expression je textový reťazec, ktorý definuje požiadavky na vyfiltrovanie vstupných dát



Obrázok 1: Funkcia Progress graph. Na ľavej strane je vybraný konkrétny dátum. Na pravej strane je aktuálny dátum.

Účelom funkcie Progress graph nie je len vypísanie počtu uzlov k danému dátumu, ale aj upravenie zobrazovanej kolekcie vzhľadom k tomuto dátumu. To znamená, že keď užívateľ klikne na nejaký dátum, tak sa zobrazovaná kolekcia zredukuje na uzly, ktoré v tom dátume boli súčasťou kolekcie. Fungovať to bude ako návrat v čase. Samozrejmosťou je aj možnosť kliknutím sa vrátiť do súčasnosti a zobraziť tak celú kolekciu.

Prepínanie rokov, umiestnené v hornej časti, bude reprezentované troma nápismi (rokmi). Prostredný rok bude aktuálne vybraný. Naľavo od neho bude minulý rok a analogicky na pravej strane bude budúci rok. V prípade, že je aktuálny rok posledný v kolekcií a žiadny budúci nie je, tak sa na pravej strane zobrazí text „[now]", po kliknutí naň sa nastaví aktuálny dátum a obnoví sa celá kolekcia. Ak je aktuálny rok prvým rokom v kolekcií, tak sa na ľavej strane nezobrazí nič. V prepínaní budú dostupné len roky, ktoré sa vyskytujú v kolekcií. Napríklad: ak v roku 2005 nepribudol žiadny uzol do kolekcie, tak po roku 2004 bude nasledovať rok 2006. Je to kvôli tomu, že v roku 2005 nenastala žiadna zmena a teda je zbytočné ho tam zahŕňať.

Pri prechádzaní kurzorom myši ponad graf sa stĺpcu pod ním zmení farba na červeno. Žltou farbou bude zvýraznený stĺpec s aktuálne vybraným dátumom. V spodnej časti bude zobrazený aktuálne vybraný dátum a počet uzlov k tomuto dátumu.

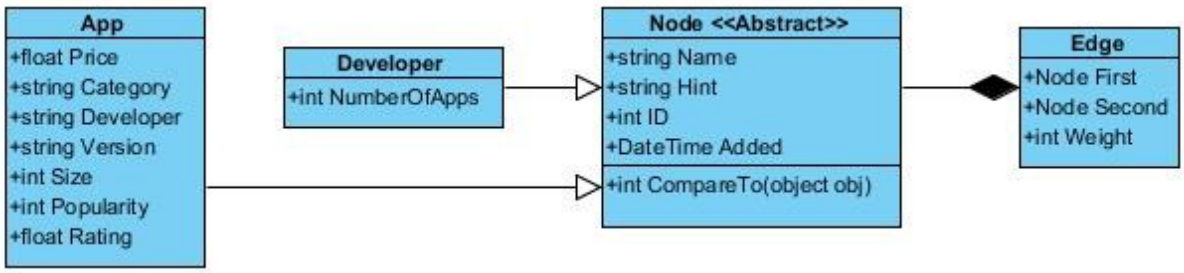
3.2 Návrh dátovej štruktúry

V tejto podkapitole sa budeme zaoberať dátovou štruktúrou komponenty. Popíšeme obsah tried slúžiacich na uchovanie dát, aj pomocných tried zabezpečujúcich funkcionalitu. Tiež popíšeme zloženie vstupného xml súboru.

Model
Hlavná trieda, ktorá reprezentuje celú dátovú vrstvu. Prezentačná vrstva obsahuje jednu inštanciu tejto triedy, cez ktorú vykonáva všetko potrebné na prácu s dátami. Trieda Model obsahuje zoznam uzlov (Node), v ktorom má uložené všetky uzly načítanej kolekcie a tiež zoznam hrán (Edge), v ktorom sú uložené hrany medzi týmito uzlami. Jej súčasťou je aj zoznam rokov (Year), do ktorého sa ukladajú zotriedené referencie na uzly pre potreby funkcie Progress graph. Za zmienku ešte stojí inštancia triedy Subgraphs, ktorá zabezpečuje funkcionalitu ohľadom podgrafov.

Keďže trieda Model reprezentuje celú dátovú vrstvu, tak obsahuje aj všetky potrebné metódy, ktoré volá prezentačná vrstva, čím zabezpečuje načítanie dát z xml, vyhľadávanie, prípravu dát pre Progress graph, ...

3.2.1 Triedy kolekcie



Obrázok 2: Class diagram tried kolekcie

Node
Táto trieda je abstraktná a slúži len na predpis najnutnejších požiadaviek na jeden uzol. Z nej sú odvodené konkrétne triedy slúžiace na uchovanie dát, ktoré si vytvorí programátor podľa potrieb danej kolekcie. Predpisuje názov uzla, pomocný reťazec (slúžiaci na rozličné pomocné dáta, ale nemusí sa využívať), číslo ID a dátum s časom pridania do kolekcie. Tento dátum môže slúžiť ako dátum vydania, narodenia a podobne, pričom záleží len na type kolekcie, aký dátum sa tam bude ukladať. Podstatné je, že tento dátum je potom kľúčový pre funkciu Progress graph.

Vo výslednej aplikácií budeme používať ukážkovú kolekciu s tvorcami na mobilnú platformu iOS a ich aplikáciami, takže ich použijeme na popísanie dátových tried odvodených z Node. Samozrejme, že tieto sa budú meniť a každý programátor si ich vytvorí sám podľa vlastnej potreby. V našom prípade sú to triedy App a Developer. Táto kolekcia pochádza z portálu AppShopper[10].

App
Táto trieda je odvodená z triedy Node a rozširuje ju o potrebné údaje o danej aplikácií. Jedná sa o tieto údaje: cena, kategória, názov vývojára, označenie verzie, veľkosť v kB, popularita aplikácie a jej hodnotenie užívateľmi. Cena a hodnotenie sú typu float, pretože obsahujú reálne čísla. Kategória, vývojár, verzia sú typu string a veľkosť s popularitou sú typu integer. Všetky ostatné údaje ako dátum pridania alebo názov aplikácie pokryje trieda Node.

Developer
Táto trieda je tiež odvodená z Node a rozširuje ju len o počet aplikácií, ktoré má daný vývojár.

Edge
Slúži na uchovanie hrany medzi dvoma uzlami. Obsahuje dve referencie na uzly, ktoré spája a číslo označujúce váhu tohto spojenia. Táto váha určuje hrúbku čiary medzi dvoma uzlami a tiež sa dá nastaviť zobrazovanie hrán od určitej váhy.

[10] AppShopper – portál, ktorý sa zaoberá aplikáciami a hrami na mobilnú platformu iOS a operačný systém Mac OS X (web: www.appshopper.com)

3.2.2 Štruktúra vstupného xml súboru

Vstupný xml súbor má jednoduchú štruktúru, kde názov koreňového elementu je „data“. Tento element obsahuje tri hlavné elementy: developers, apps a edges. Každý z nich obsahuje jednotlivé uzly/hrany.

Napríklad element developers obsahuje všetkých vývojárov a každý je reprezentovaný elementom „developer“ s atribútmi: id, name, count a date. Podobne aj apps obsahuje jednotlivých vývojárov ako elementy „app“ s príslušnými atribútmi.

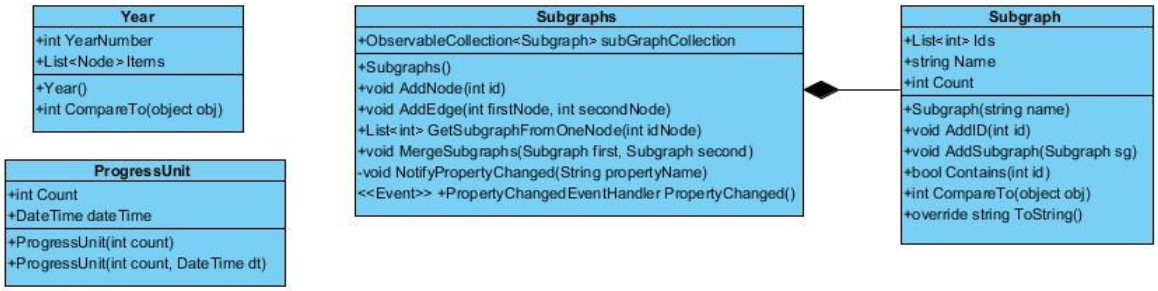
A na záver, element edges obsahuje jednotlivé hrany ako elementy „edge“ s atribútmi: first, second, weight, kde first a second obsahujú id čísla uzlov a weight obsahuje váhu hrany.

Pochopiteľne štruktúru xml si určí programátor a bude pre každý typ kolekcie unikátna. Podľa určenej štruktúry treba upraviť aj metódu LoadData() z triedy Model, aby predpokladala takto štruktúrovaný vstup. Naša štruktúra počíta s dvoma typmi uzlov: App a Developer. Samozrejme záleží len na programátorovi, koľko typov si zvolí. Jedinou podmienkou je, aby ich ID bolo unikátne, keďže sú to vo výsledku všetko uzly. Nemôže sa stať, aby mali dva uzly rovnaké ID aj keď je jeden z nich typu App a druhý typu Developer, pretože potom by nastali komplikácie pri vytváraní hrán. Hrany sú na vstupe totiž adresované práve pomocou týchto ID čísiel. Až po načítaní budú adresované referenciami na vytvorené objekty.

Ukážka vstupného xml súboru:

```
1.  <data>
2.
3.   <developers>
4.    <developer id='0' name='Rovio Mobile Ltd.' count='16' date='2010-10-20' />
5.   </developers>
6.
7.   <apps>
8.    <app id='1' name='Angry Birds' category='Games' developer='Rovio Mobile Ltd.'
    popularity='86' price='3.99' size='43900' rating='3.00' version='2.0.2' date='20
    11-01-07' />
9.    <app id='2' name='Angry Birds Free' category='Games' developer='Rovio Mobile L
    td.' popularity='144360' price='0.0' size='17600' rating='4.00' version='1.3.1'
    date='2011-01-06' />
10.  </apps>
11.
12.  <edges>
13.   <edge first='0' second='1' weight='4' />
14.   <edge first='0' second='2' weight='0' />
15.  </edges>
16.
17. </data>
18.
```

3.2.3 Pomocné triedy



Obrázok 3: Class diagram pomocných tried

Subgraph
Táto trieda reprezentuje jeden podgraf. Obsahuje zoznam ID čísel uzlov a názov podgrafu. Okrem toho obsahuje aj metódy na pridanie jedného nového ID, pridanie všetkých ID čísiel z iného podgrafu, zistenie prítomnosti konkrétneho ID čísla a zistenie počtu všetkých ID.

Subgraphs
Táto trieda reprezentuje celú funkcionalitu ohľadom podgrafov. Model iba posiela uzly a hrany tejto triede a tá ich automaticky zaraďuje do jednotlivých podgrafov podľa hrán. Obsahuje kolekciu inštancií triedy Subgraph a obslužné metódy. Taktiež obsahuje aj potrebnú funkcionalitu pre DataBinding technológie Silverlight. Ten sa využíva pri vypisovaní podgrafov v prezentačnej vrstve a vďaka nemu sa dá tento výpis veľmi flexibilne prispôsobovať.

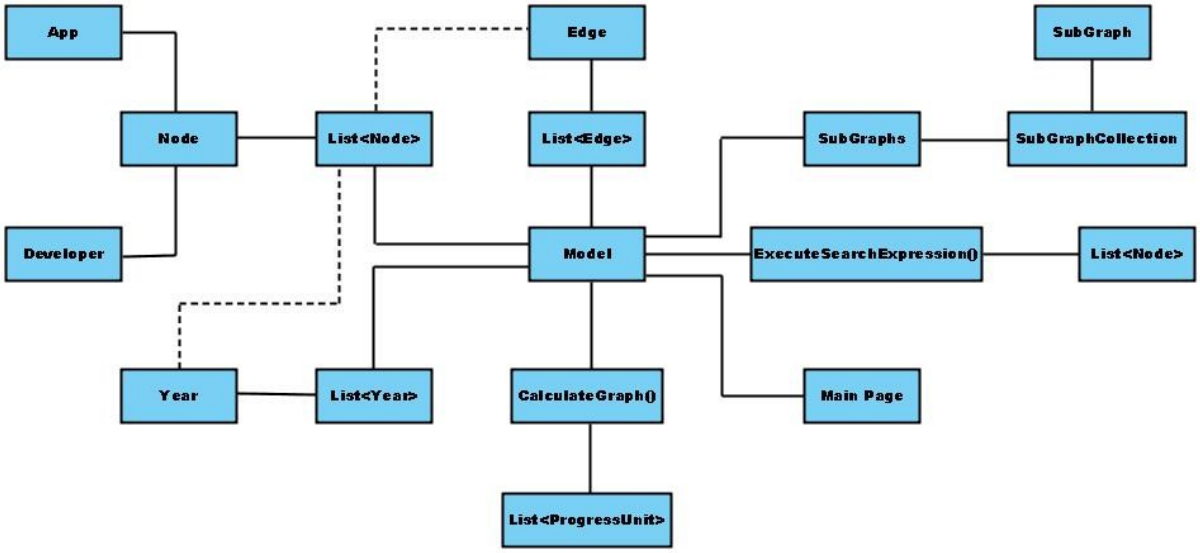
ProgressUnit
Táto trieda predstavuje jeden konkrétny dátum pre funkciu Progress graph. Obsahuje počet uzlov a konkrétny dátum. Vo vizuálnej časti je reprezentovaná ako jeden stĺpec, na ktorý je možné kliknúť pre výber dátumu.

Year
Táto trieda predstavuje jeden rok vo funkcií Progress graph. Obsahuje číslo roku a pole referencií na uzly, ktoré je chronologicky zoradené.

3.3 Návrh architektúry

Návrh architektúry tejto komponenty sa skladá z dvoch hlavných častí, ktoré sú vzájomne oddelené a do určitej miery nezávislé. Tie dve časti sú dátová a prezentačná vrstva. Každá je v samostatnom projekte toho istého riešenia (Solution). Dátová vrstva je reprezentovaná triedou Model a prezentačná vrstva je reprezentovaná triedou MainPage. Ako už bolo spomínané, trieda Model má na starosti všetko ohľadom spracovania dát. Na druhú stranu, trieda MainPage sa stará o vizualizáciu komponenty. V nej je všetko potrebné na zobrazenie grafického užívateľského rozhrania, vykresľovanie uzlov/hrán a interakciu s nimi. Keďže je to silverlight aplikácia, tak má oddelený grafický dizajn od riadiacej logiky programu, čo je veľmi praktické pri upravovaní vzhľadu komponenty, pretože ho môžeme ľubovoľne upravovať bez toho, aby to narušilo funkčnosť programu.

V tejto podkapitole popíšeme architektúru komponenty, jej najvýznamnejšie triedy a vzťahy medzi nimi. Nebudeme popisovať všetky triedy, pretože mnohé z nich nie sú dostatočne významné pre fungovanie komponenty. Zameriame sa iba na tie, ktoré sú potrebné pre pochopenie fungovania celého mechanizmu.



Obrázok 4: Návrh architektúry komponenty

Na obrázku 4 je vyobrazený návrh architektúry komponenty so zameraním na funkčnosť dátovej vrstvy. Ako už bolo spomenuté, celé spracovanie dát sa sústredí okolo triedy Model a je to aj prehľadne znázornené na obrázku. Pre vysvetlenie fungovania celej komponenty musíme začať najskôr u triedy MainPage, ktorá inicializuje celú komponentu. Táto trieda je po štarte načítaná ako prvá a má za úlohu vytvoriť grafické užívateľské rozhranie a jednu inštanciu triedy Model. Následne čaká na vstup od užívateľa, ktorý vloží vstupné xml s dátami. Trieda MainPage tento xml súbor otvorí a jeho obsah odovzdá triede Model.

V nej prebehne extrahovanie dát, počas ktorého sa naplnia tri zoznamy: uzly, hrany a roky. Tieto tri zoznamy sú v schéme zobrazené ako: List<Node>, List<Edge> a List<Year>. Zoznam uzlov obsahuje všetky inštancie uzlov s dátami. Tie sú uchované v triedach odvodených z abstraktnej triedy Node, v prípade našej implementácie: App a Developer. Zoznam hrán obsahuje inštancie triedy Edge, ktorá si pamätá váhu spojenia a dva spájané uzly, ktoré určuje odkazovaním do zoznamu uzlov (v schéme znázornené prerušovanou čiarou). Zoznam rokov obsahuje inštancie triedy Year, ktorá sa tiež odkazuje do zoznamu uzlov, pretože si uchováva odkazy na všetky uzly pridané v danom roku.

Počas vytvárania jednotlivých uzlov/hrán si trieda Model zároveň aj tieto uzly/hrany pridáva do triedy SubGraphs. Tá zabezpečuje funkcionalitu zoskupovania do podgrafov. Obsahuje kolekciu nazvanú SubGraphCollection, ktorá obsahuje jednotlivé inštancie triedy SubGraph a sprostredkuje DataBinding pre triedu MainPage.

Po vykonaní týchto úkonov je celá vstupná kolekcia načítaná a spracovaná. Vykonávanie sa opäť presunie do triedy MainPage. Tá si z triedy Model načíta zoznam uzlov a pre každý jeden uložený uzol vytvorí jeho vizuálnu reprezentáciu v podobe kruhu. To isté spraví aj so zoznamom hrán a z jeho záznamov vytvorí prepojenia medzi týmito kruhmi. Počas toho upraví ich váhy koeficientom podľa najvyššej zaznamenanej hrany, tak aby najvyššia zaznamenaná hrana predstavovala 100 percent. Potom do tabuľky podgrafov nastaví odkaz na SubGraphCollection, ktorá túto tabuľku automaticky naplní pomocou funkcie DataBinding. Následne je zavolaný algoritmus na rozmiestnenie uzlov a metóda na nastavenie veľkostí uzlov podľa ich významnosti. Tu je samozrejme možné si nastaviť podľa ľubovoľných kritérií.

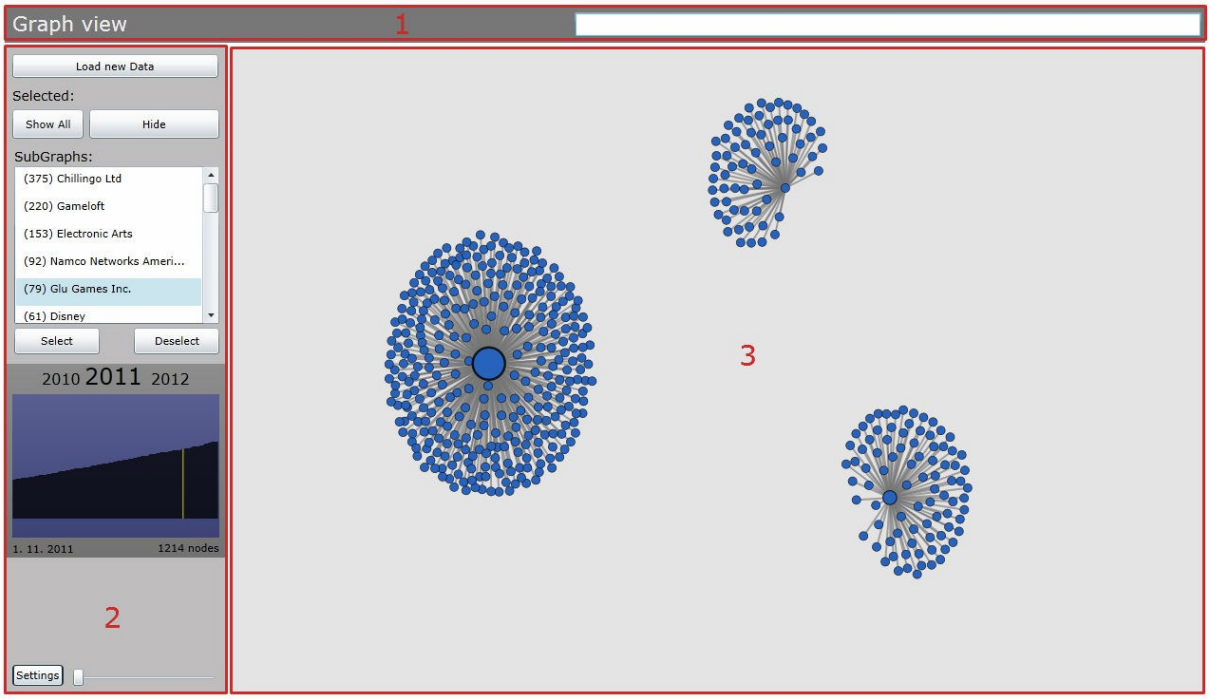
Na záver zobrazí výslednú grafovú štruktúru na canvas[11] a zistí, aký je posledný rok v kolekcii. Ten použije ako vstup metódy DrawProgressYear, ktorá slúži na vykreslenie vývoja v konkrétnom roku. K tomu používa jednu metódu z triedy Model, ktorá je znázornená na obrázku 4 a slúži na prepočítanie vývoja v danom roku. Táto metóda sa volá CalculateGraph a jej funkčnosť bude vysvetlená v kapitole 4. Zatiaľ je podstatné len to, že požaduje číslo roku a šírku vykresľovanej plochy, aby vedela na koľko stĺpcov má prepočítať kroky. Návratovou hodnotou je zoznam inštancií triedy ProgressUnit a tento zoznam je následne použitý na vykreslenie jednotlivých stĺpcov.

Po vykonaní týchto úkonov je komponenta pripravená na používanie. Je možné s ňou plne pracovať, užívateľ môže manipulovať s jednotlivými uzlami, filtrovať ich, odoberať z kolekcie a podobne, čím je pokrytá všetka funkcionalita, ktorá bola popísaná v kapitole 3.1 Špecifikácia funkcionality. Je možné vykonávať aj vyhľadávanie pomocou vyhľadávacieho riadku (Search bar), ktorý je súčasťou GUI. Keď do neho užívateľ zadá požadovaný Query Expression a potvrdí ho, je tento reťazec použitý ako vstup metódy ExecuteSearchExpression z triedy Model (viz. Obrázok 4). Táto metóda vráti zoznam uzlov, ktoré vyhovujú zadaným podmienkam. Následne sú tieto uzly vybrané a užívateľ s nimi môže pracovať.

3.4 Návrh GUI

Grafické užívateľské rozhranie je navrhnuté s ohľadom na jednoduchosť a prehľadnosť. Farebné zladenie je modro-šedé s kontrastnou farbou oranžovou a všetky farby sú v menej saturovaných odtieňoch. Celé GUI je rozdelené na tri časti: vrchný riadok, bočná lišta a canvas.

[11] Canvas – plocha určená na vykresľovanie



Obrázok 5: GUI a jeho rozdelenie

Popis:

a. Vrchný riadok – nachádza sa tu názov a vyhľadávací riadok

b. Bočná lišta – je určená na ovládacie prvky (s výnimkou kontextového menu), sú tu umiestnené tlačítka, ovládanie podgrafov, Progress graph, ovládanie limitu váhy hrán a tlačítko pre nastavenie.

c. Canvas – vykresľovacia plocha zaberá pochopiteľne najviac miesta a slúži na vizualizáciu grafu. Práve tu prebieha celé manipulovanie s kolekciou.

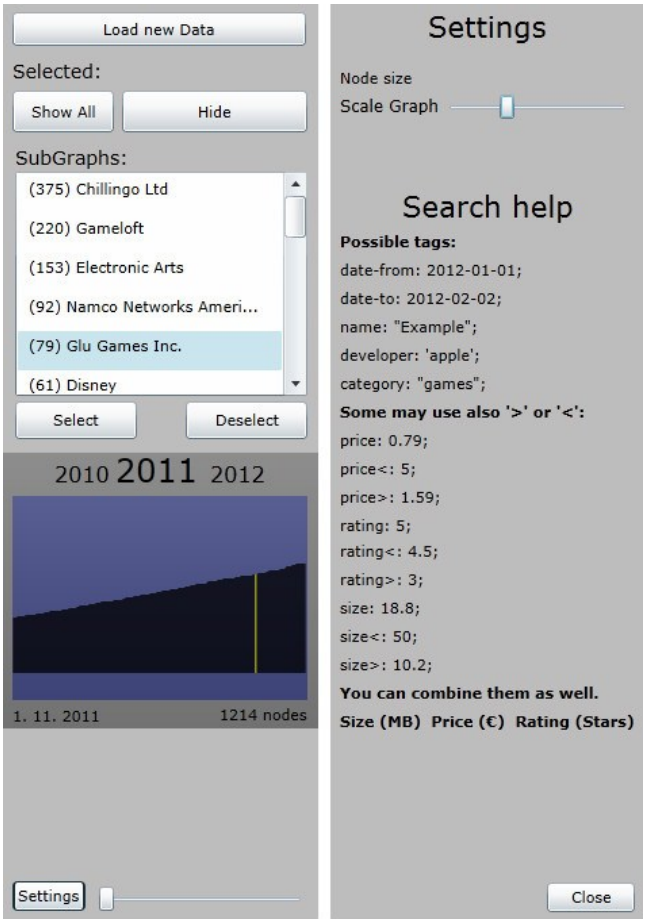
3.4.1 Bočná lišta

Bočná lišta je miesto, kam patria všetky ovládacie prvky. Je iba na programátorovi, ako si ju navrhne, prípadne čo všetko sem umiestni. Je tu umiestnená tabuľka s podgrafmi. Ku každému podgrafu je napísaný počet uzlov v ňom a jeho názov. Je možné ich vyberať a pomocou tlačidiel pod tabuľkou nechať vybrať všetky jeho uzly.

Nachádza sa tu aj Progress graph. Ten má v hornej časti vypísané tri roky, kde prostredný je aktuálne vybraný a je zobrazený väčším písmom. Zvyšné dva slúžia na prepínanie rokov dopredu a dozadu. Ako už bolo spomenuté v kapitole 3.1, v prípade, že je aktuálny rok posledným v kolekcií, tak je namiesto tretieho roku nápis „Now“, ktorý nastaví dátum na súčasnosť. Pod týmito rokmi sa nachádza samotné vykreslenie vývoja. Je zložené zo stĺpcov, kde každý z nich má šírku 1 pixel a predstavuje jeden konkrétny dátum. Taktiež majú všetky stĺpce nastavené udalosti na obsluhovanie zvýraznenia pod kurzorom myši a kliknutie pre vybranie ich dátumu. Vždy je zobrazené iba rozpätie jedného roku. Všetky mesiace v roku majú rovnakú šírku v pixeloch a teda

rovnaký počet stĺpcov. Mení sa len časové obdobie medzi dvoma stĺpcami, pretože všetky mesiace nemajú rovnaký počet dní. Toto časové rozpätie je konštantné v rámci jedného mesiaca a okrem počtu dní v mesiaci je závislé na celkovej šírke vykresľovanej plochy. Dopodrobna popíšeme výpočet tohto časového rozpätia v kapitole 4. Výška stĺpcov je tiež prepočítaná vzhľadom k rozmerom vykresľovanej plochy proporcionálne tak, aby mal najnižší stĺpec výšku aspoň 50 pixelov. Toto zabezpečuje prehľadnosť a ľahké klikanie na daný stĺpec. Po kliknutí na nejaký stĺpec sa zmena dátumu okamžite prejaví na canvase a výber je možné úplne zrušiť len kliknutím na nápis „Now“. Pod zobrazením vývoja sa nachádza informácia o aktuálne vybranom dátume a počte uzlov k tomuto dátumu.

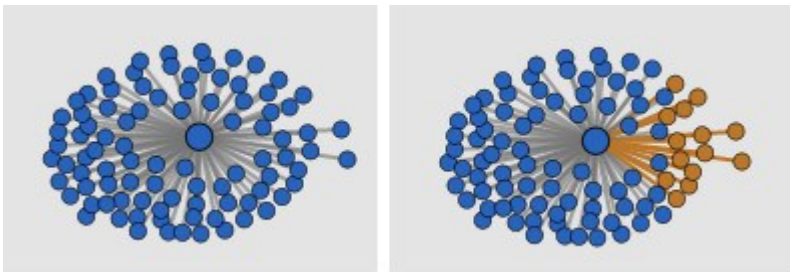
Na spodnej strane sa nachádza tlačítko pre nastavenia a posuvník. Tento posuvník slúži na nastavenie spodnej hranice váhy hrán. To znamená, že budú zobrazované len hrany, ktoré majú váhu vyššiu, ako je táto hranica. Po stlačení tlačítka pre nastavenie sa celá bočná lišta zmení na nastavenie. Opäť je len na programátorovi, aké nastavenia sem umiestni. Táto lišta je vhodná napríklad aj na nápovedu. V našej implementácií sa tu nachádza zmena mierky veľkosti uzlov a nápoveda k vyhľadávaciemu riadku.



Obrázok 6: Bočná lišta v normálnom režime (vľavo) a v režime nastavení (vpravo)

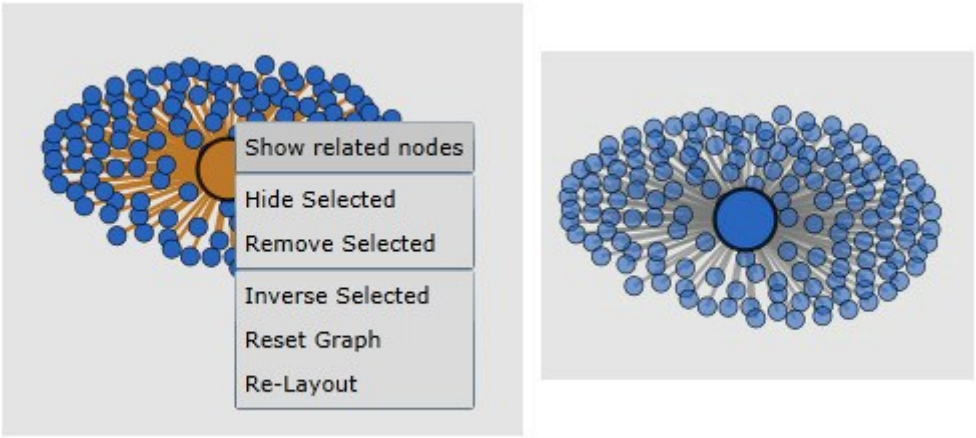
3.4.2 Canvas

Na canvase sa zobrazuje celá kolekcia. Uzly sú zobrazované modrou farbou a majú rozličné veľkosti podľa významu, ktorého kritéria si určí programátor. Vyselektované uzly majú oranžovú farbu.



Obrázok 7: Ukážka vykresľovania uzlov a hrán. Na pravej strane je ukážka výberu určitých uzlov

Po kliknutí pravým tlačítkom myši sa zobrazí kontextové menu. Jeho obsah sa mení v závislosti na tom, či užívateľ klikne na nejaký uzol alebo nie a tiež podľa toho, či sú vybrané nejaké uzly. Toto kontextové menu ponúka možnosti ako zobrazenie okolitých uzlov, schovanie výberu, odobratie z kolekcie, zobrazenie neviditeľných uzlov, obrátenie výberu, obnova grafu a znovu usporiadanie uzlov.



Obrázok 8: Ukážka kontextového menu (vľavo) a zobrazenie okolitých uzlov (vpravo)

4. Implementácia

V tejto kapitole popíšeme výslednú implementáciu našej komponenty. Do detailu popíšeme a vysvetlíme fungovanie metód CalculateGraph, GetNodesByDateRange a ExecuteSearchExpression. Kvôli prehľadnosti budeme tieto metódy popisovať po častiach. Na záver tejto kapitoly uvedieme výsledky výkonnostných testov našej komponenty.

4.1 Metóda CalculateGraph

Táto metóda požaduje dva parametre typu integer. Prvým je číslo roku, ktorý má spočítať a druhý parameter je šírka vykresľovanej plochy pre Progress graph. Po vykonaní celého algoritmu vracia táto metóda List<ProgressUnit>, v ktorom sú uložené počty uzlov pre konkrétne dátumy.

```
1.  public List<ProgressUnit> CalculateGraph(int year, int width)
2.  {
3.      List<ProgressUnit> tmp = new List<ProgressUnit>();
4.      int y = -1; // position of this year in list
5.      int count = 0;
6.      int mw = (width / 12); // width in pixels for one month
7.      DateTime current = new DateTime(year, 1, 1);
8.      double ms = 0.0; // Month-Step -
     step inside particular month, it's based on number of days in current month and
     "MW"
9.      int id = 0; // iterator
10.
11.     // BASE COUNT - sum of all the years before, it's a starting value
12.     for (int i = 0; i < Years.Count; i++)
13.     {
14.         // looking for choosen year
15.         if (Years[i].YearNumber == year)
16.             y = i;
17.
18.         // actual counting
19.         if (Years[i].YearNumber < year)
20.         {
21.             count += Years[i].Items.Count;
22.         }
23.         else if (Years[i].YearNumber > year)
24.             break;
25.     }
26.
27.     // In case that choosen year was not found
28.     if (y == -1)
29.     {
30.         tmp.Add(new ProgressUnit(count));
31.         return tmp;
32.     }
```

Na začiatku metódy sa vytvoria všetky potrebné premenné. Premenná „tmp“ je prázdny zoznam inštancií triedy ProgressUnit a slúži ako návratová hodnota tejto metódy. Premenná „y“ bude v sebe uchovávať pozíciu zvoleného roku v zozname všetkých rokov. Do premennej „count“ sa

postupne pripočítavajú uzly a tak sa pomocou nej získavajú hodnoty pre jednotlivé inštancie ProgressUnit. Premenná „mw“ (Month Width – šírka mesiaca) obsahuje šírku v pixeloch pre jeden mesiac. „Current“ bude slúžiť na iterovanie v dátumoch po jednotlivých krokoch. Premenná „ms“ (Month-Step – krok v mesiaci) uchováva počet dní v desatinnom tvare o koľko sa treba posunúť na jeden krok (jeden stĺpec). To zabezpečuje, aby boli časové rozstupy medzi jednotlivými stĺpcami konštantné pre celý mesiac. Ako posledná je premenná „id“, ktorá slúži na iterovanie v uzloch. Pomocou nej sa postupne posúvame v chronologicky zoradenom zozname a počítame, koľko uzlov už v danom čase existovalo.

Potom príde na rad vypočítanie tzv. Base count[12]. For cyklom sa prechádza celý zoznam rokov a overuje sa, či je číslo roku menšie, rovné alebo väčšie ako požadovaný rok. V prípade, že je menšie, tak sa do premennej „count“ pripočíta počet všetkých uzlov v tomto roku. Ak je číslo roku rovné požadovanému, tak sa do premennej „y“ priradí zistená pozícia požadovaného roku v tomto zozname. V prípade, že je číslo roku väčšie, tak sa ukončí prechádzanie tohto zoznamu. V tomto bode už vieme, na ktorej pozícii v zozname sa nachádza požadovaný rok a tiež vieme, koľko uzlov bolo v kolekcií pred začiatkom roku.

Nasleduje poistka pre prípad, že by požadovaný rok nebol nájdený v kolekcií. K takému stavu by nemalo dôjsť, pretože naša implementácia tejto funkcie nedovoľuje vybrať rok, ktorý sa v kolekcií nenachádza, pretože pri prepínaní rokov ignoruje neexistujúce roky (také, ktoré nemajú záznam). Na druhú stranu, iný programátor môže požadovať prepínanie aj medzi takýmito rokmi a preto je tu táto poistka vytvorená. V takom prípade je vrátený zoznam s jedným záznamom obsahujúcim počet uzlov, ktorý sa pochopiteľne celý rok nezmení.

Príprava je hotová, teraz nasleduje samotný výpočet jednotlivých krokov:

```
34. // runs every month
35.     for (int m = 0; m < 12; m++)
36.     {
37.         ms = (double)DateTime.DaysInMonth(year, m+1) / mw; // number of days div
    ided by pixels for one month
38.
39.         // runs every month-step
40.         for (int i = 0; i < mw; i++)
41.         {
42.             current = current.AddDays(ms); // move comparing datetime forward
43.
44.             // Count every node that is
45.             if (id < Years[y].Items.Count)  // protection
46.                 while (Years[y].Items[id].Added.CompareTo(current) <= 0)
47.                 {
48.                     count++; // count this node
49.
50.                     if (++id == Years[y].Items.Count) // and move on the next
51.                         break;
52.                 }
53.
54.             tmp.Add(new ProgressUnit(count, current));
```

[12] Base count (základný počet) - štartovacia hodnota. Označuje počet uzlov v kolekcií pred začiatkom zvoleného roku.

```
55.         }
56.     }
57.
58.         return tmp;
59. }
```

Nasledovný algoritmus sa vykoná dvanásťkrát. Keďže každý mesiac má iný počet dní, tak sa musí pre neho vypočítať zvlášť krok v mesiaci. Platí, že 1 pixel = 1 stĺpec = 1 krok v mesiaci, takže stačí iba predeliť počet dní v danom mesiaci počtom pixelov na vykreslenie celého mesiaca (viz. riadok 37).

Z vyššie uvedenej rovnice vyplýva, že počet pixelov pre jeden mesiac sa rovná počtu stĺpcov/krokov. Preto nasleduje for cyklus, ktorý sa vykoná toľko krát, akú hodnotu má premenná „mw“. V každej iterácií tohto cyklu sa posunie dátum v premennej „current“, tým sa posunieme o jeden krok. Na riadku 45 je ochrana proti stavu, kedy by sme sa posúvali ďalej, ale už by neostali žiadne nové uzly v tomto roku. Pokým ešte sú v danom roku nejaké uzly, ktoré sme nezapočítali, tak sa posúvame ďalej v zozname uzlov od posledného započítaného uzla a kontrolujeme, či je jeho dátum skorší než dátum v premennej „current“. Ak áno, tak navýšime premenné „count“ a „id“. Overíme si, či sme už nevyčerpali všetky uzly v roku, ak áno, tak už ďalšie uzly nekontrolujeme. Nakoniec jedného kroku sa vytvorí nová inštancia triedy ProgressUnit, nastaví sa jej dátum z premennej „current“ a počet uzlov k tomuto dátumu z premennej „count“. Táto inštancia sa potom pridá do „tmp“ a celý cyklus ide od začiatku. Opäť sa overovací dátum posunie o konštantný počet dní a overia sa uzly pridané do kolekcie skôr než tento dátum. Po vykonaní všetkých krokov sa posunieme na ďalší mesiac a celé toto krokovanie začne odznova. Na konci metódy sa vráti zoznam „tmp“.

4.2 Metóda GetNodesByDateRange

Táto metóda nie je síce až tak významná ako zvyšné dve, ktoré sme sa rozhodli popísať, ale je využívaná jednou z nich. Konkrétne metódou ExecuteSearchExpression, ktorú budeme popisovať v ďalšej podkapitole a tak sme sa rozhodli pre úplnosť popísať aj túto menšiu metódu. Jej účel je jednoduchý, na vstupe požaduje dva dátumy (trieda DateTime) a návratová hodnota je zoznam uzlov, ktoré sa nachádzajú v tomto časovom rozpätí.

```
1.  public List<Node> GetNodesByDateRange(DateTime first, DateTime second)
2.  {
3.      DateTime f, s;
4.      List<Node> tmp = new List<Node>();
5.
6.      // they could be in a non-chronologic order (second is earlier than first)
7.      if (first.CompareTo(second) < 0)
8.      {
9.          f = first;
10.         s = second;
11.     }
12.     else
13.     {
14.         f = second;
15.         s = first;
16.     }
```

Na úvod sa pripravia pomocné premenné. Konkrétne prázdny zoznam uzlov, ktorý bude slúžiť ako návratová hodnota a dve referencie na DateTime s názvami „f“ a „s“. Keďže zadané dátumy nemusia byť v chronologickom poradí (first môže byť neskorší dátum ako second), tak sa overí ich poradie a referencie „f“ a „s“ sa priradia tak, aby „f“ bolo skôr ako „s“.

```
18. foreach (Year y in Years)
19.         if (y.YearNumber < f.Year || y.YearNumber > s.Year)  // years outside ra
    nge are simply excluded
20.         {
21.             continue;
22.         }
23.         else if (y.YearNumber > f.Year && y.YearNumber < s.Year) // years inside
    , but not boundary ones, are included
24.         {
25.             foreach (Node n in y.Items)
26.                 tmp.Add(n);
27.         }
28.         else // there are only boundary years (those, which are set)
29.         {
30.             foreach (Node n in y.Items)
31.                 if ((n.Added.CompareTo(f) >= 0) && (n.Added.CompareTo(s) <= 0))

32.                     tmp.Add(n);
33.         }
34.
35.     return tmp;
36. }
```

Keď už sú v pomocných referenciách správne zoradené vstupné dátumy, môže sa vykonať porovnávanie. To sa skladá z jedného foreach cyklu, ktorý prechádza všetkými rokmi. Namiesto náročného porovnávania všetkých uzlov, ich filtrujeme podľa roku. Môžu nastať tri stavy. Ak je kontrolovaný rok menší ako „f“, alebo je tento rok väčší ako „s“, tak je zrejmé, že všetky jeho uzly budú mimo požadovaný rozsah a preto sa daný rok automaticky ignoruje. Druhý možný stav je, že kontrolovaný rok, je medzi zadanými dvoma, ale nie je zhodný ani s jedným z nich. V takom prípade je jasné, že všetky jeho uzly budú vyhovovať rozsahu a sú automaticky pridané do výsledného zoznamu.

Tretí možný stav je, že kontrolovaný rok je jeden zo zadaných rokov. V takom prípade sa už nedajú odfiltrovať na základe roku, a musia sa porovnávať po jednom s presnosťou na deň. Teoreticky sa porovnávajú s presnosťou na milisekundy, pretože trieda DateTime uchováva nielen dátum, ale aj čas. Naša implementácia však na vstupe požaduje iba dátumy, takže čas je irelevantný. Samozrejme nie je problém si v kolekcií uchovávať konkrétny dátum aj s časom, je to len na programátorovi, ako si bude túto komponentu implementovať. Táto metóda je pripravená aj na taký prípad.

Toto porovnávanie prechádza všetkými uzlami v danom roku a každý z nich kontroluje pomocou metódy CompareTo z triedy DateTime, či je neskorší ako „f“ a zároveň skorší ako „s“. Ak uzol prejde kontrolou, tak je pridaný do zoznamu a na záver je vrátený celý zoznam.

4.3 Metóda ExecuteSearchExpression

Táto metóda slúži na spracovanie zadaných podmienok vyhľadávania. Na vstupe požaduje iba string z vyhľadávacieho riadku a návratová hodnota je zoznam uzlov, ktoré vyhovujú zadaným podmienkam. Každý programátor si pri implementovaní tejto komponenty určí svoje vlastné kritéria vyhľadávania podľa potrieb danej dátovej kolekcie, ktorú bude zobrazovať. V prípade, že sa vyskytne nejaká chyba alebo je daný výraz neplatný, tak sa vracia null. Nebudeme tu uvádzať celý zdrojový kód, pretože je dlhý a niektoré časti sa opakujú. Pre pochopenie postačí, ak popíšeme všetko, čo sa neopakuje.

```
1.  public List<Node> ExecuteSearchExpression(string expression)
2.  {
3.      if (expression == "" || Years.Count == 0 || Years[0].Items.Count == 0)
4.          return null;
5.
6.      List<Node> result = new List<Node>();
7.      string tag, command, value;
8.      bool bName = false, bPrice = false, bPriceH = false, bPriceL = false, bRatin
    g = false, bRatingH = false, bRatingL = false,
9.           bCategory = false, bDeveloper = false, bSize = false, bSizeH = false, b
    SizeL = false;
10.
11.     DateTime firstDT = Years[0].Items[0].Added; // The most oldest date
12.     DateTime secondDT = DateTime.Now;
13.     string name = "", category = "", developer = "";
14.     float price = 0.0f, priceH = 0.0f, priceL = 0.0f;
15.     float rating = 0.0f, ratingH = 0.0f, ratingL = 0.0f;
16.     float size = 0.0f, sizeH = 0.0f, sizeL = 0.0f;
```

Na začiatku sa overí, či hľadaný reťazec nie je prázdny, či sú načítané nejaké roky a či prvý rok obsahuje nejaké uzly. Ak jedna z týchto podmienok nie je splnená, tak sa vráti null. Potom sa pripravia pomocné premenné: zoznam uzlov, ktorý slúži ako návratová hodnota, pomocné stringy na extrahovanie tagov (kritérií) a hodnôt. Nasleduje séria premenných typu boolean, kde má každé z kritérií svoju vlastnú premennú. V prípade, že je kritérium číselné, tak má až tri, pre každý stav jednu: menšie, rovné, väčšie. Do týchto premenných sa bude nastavovať, či je dané kritérium požadované alebo sa má ignorovať. Potom sa vytvoria dva dátumy a do prvého z nich sa nastaví najstarší dátum v kolekcií (chronologicky prvý uzol) a do druhého dátumu sa nastaví dnešný dátum, čo odpovedá najneskoršiemu dátumu. Ďalej pokračujú pomocné premenné, ktoré majú rozličné dátové typy v závislosti na charaktere kritérií a ukladajú sa do nich požadované hodnoty. Každé kritérium má svoju premennú, v prípade číselných kritérií má opäť tri premenné.

```
18.     // To process query
19.     while (expression.Trim().Length > 0)
20.     {
21.         try
22.         {
23.             // Extracting command and value
24.             tag = GetNextQueryTag(ref expression);
25.             command = GetTagCommand(tag);
26.             value = GetTagValue(tag);
27.
```

```
28.             // Setting appropriate options
29.             switch (command.ToLower())
30.             {
31.                 case "date-from":
32.                     firstDT = Convert.ToDateTime(value);
33.                     break;
34.
35.                 case "date-to":
36.                     secondDT = Convert.ToDateTime(value);
37.                     break;
38.
39.                 case "name":
40.                     bName = true;
41.                     name = value;
42.                     break;
```

Keď sú všetky pomocné premenné hotové, začne extrahovanie požadovaných kritérií z textového reťazca. Jednotlivé tagy, oddelené bodkočiarkou, sa postupne odoberajú zo vstupného stringu, pokiaľ nie je prázdny. Analyzovanie tagu sa vykonáva v try-catch bloku pre odchytávanie syntaktických chýb. Na začiatku každého extrahovania tagu sa vstupná premenná „expression“ skráti o medzery z oboch strán. Potom sa do premennej „tag“ uloží návratová hodnota metódy GetNextQueryTag. Túto metódu sme sa vzhľadom na jej jednoduchosť rozhodli nepopisovať do detailu. Kľúčové je vedieť, že na vstupe požaduje referenciu na string, z tohto stringu vyextrahuje všetko po najbližšiu bodkočiarku (v prípade, že tam žiadna nie je, tak do konca stringu), to vráti ako návratovú hodnotu a ešte predtým odstráni tento tag zo vstupného reťazca. Takže v tomto momente je v premennej „tag“ uložený celý tag v tvare napríklad: „price: 5.0“, čo sa následne použije ako vstup ďalších dvoch jednoduchých metód: GetTagCommand a GetTagValue. Obidve požadujú na vstupe string (v našom prípade premennú „tag“) a vracajú tiež string. Prvá z nich vracia textový reťazec po dvojbodku (teda názov tagu) a druhá vracia všetko za dvojbodkou (zadanú hodnotu). Tieto dva výstupy sa uložia do premenných „command“ a „value“.

Potom nasleduje spracovanie vyextrahovaného tagu pomocou switch-u. Tu nevkladáme celý zdrojový kód, pretože sa kód pre jednotlivé kritéria opakuje. Uvádzame len také, aby sme pokryli všetky typy. Obsah premennej „command“ sa prevedie na malé písmená a pomocou switch-u sa porovná so zadanými názvami. V prípade, že sa jedná o jeden z dátumov (date-from, alebo date-to), tak sa jeho hodnota v premennej „value“ skonvertuje na DateTime a uloží sa do príslušnej premennej. Pri dátumoch nie je potreba ich „zapínať“ pomocou premennej boolean, pretože tie sa zvažujú vždy. Druhá možnosť je, že tag má typ string (name, category, developer). V takom prípade sa len požadovaný tag „zapne“ pomocou premennej boolean a nastaví sa hodnota do pomocnej premennej.

```
84.                 case "size":
85.                     bSize = true;
86.                     size = Convert.ToSingle(value.Replace(".", ","));
87.                     break;
88.
89.                 case "size>":
90.                     bSizeH = true;
91.                     sizeH = Convert.ToSingle(value.Replace(".", ","));
92.                     break;
```

```
93.
94.                 case "size<":
95.                     bSizeL = true;
96.                     sizeL = Convert.ToSingle(value.Replace(".", ","));
97.                     break;
98.             }
99.         }
100.        catch (Exception)
101.        {
102.             MessageBox.Show("Syntax error!");
103.             return null;
104.        }
105.       }
```

Treťou možnosťou je číselná hodnota. Tá má vždy tri stavy, ktoré sa nastavujú pomocou znakov „<" a „>", resp. bez nich. Každý z týchto troch stavov má vlastné pomocné premenné a princíp je rovnaký ako pri textových tagoch. Jediný rozdiel je v tom, že tu sa hodnota konvertuje na float a ešte predtým sa všetky „bodky" nahradia „čiarkami". Toto opatrenie je kvôli konvertovaniu na float, kde konvertor očakáva desatinnú čiarku.

```
107.       // Executing commands
108.       List<Node> tmp = GetNodesByDateRange(firstDT, secondDT); // date appropri
    ate nodes
109.
110.       foreach (Node n in tmp)
111.       {
112.           // Name
113.           if (bName && !(n.Name.ToLower().IndexOf(name.Trim().Replace("\"", "")
    .ToLower()) >= 0))
114.               continue;
115.
116.           // Crash prevention before retype to App
117.           if (n.GetType() != typeof(App))
118.               continue;
119.
120.           App aNode = (App)n;
121.
122.           // Category
123.          if (bCategory && !(aNode.Category.ToLower().IndexOf(category.Trim().Re
    place("\"", "").ToLower()) >= 0))
124.               continue;
```

Keď sa program dostane na riadok 108, má už všetky pomocné premenné pripravené a môže začať vyberať uzly. Pre optimalizáciu ich najskôr pretriedi podľa dátumov a uloží ich do zoznamu „tmp". Ten potom postupne prechádza a každý uzol skontroluje na všetky podmienky. V prípade, že neprejde nejakou podmienkou, tak automaticky preskakuje daný uzol a ide kontrolovať ďalší v poradí. Ak uzol prejde všetkými podmienkami, tak je na konci uložený do zoznamu „result" a ten je potom použitý ako návratová hodnota metódy.

Ako prvý sa skontroluje názov, pretože ten je spoločný pre všetky uzly (predpisuje ho abstraktná trieda Node). Najskôr sa overí boolean premenná, či je vôbec tento tag zadaný, ak áno, tak sa zadaná hodnota tagu skráti o prebytočné medzery, odstránia sa úvodzovky a prevedie sa na malé písmená. Tento upravený reťazec sa potom kontroluje na výskyt v názve uzlu, ktorý bol tiež

prevedený na malé písmená. Ak sa tam nenachádza, tak sa tento uzol preskakuje, pretože nevyhovel podmienkam.

Ak sa program dostane až sem, tak vieme, že daný uzol prešiel kontrolou na dátum a meno. Na rade sú parametre konkrétneho typu. Keďže trieda Developer nemá okrem dátumu a názvu nič, čo by sa oplatilo vyhľadávať, tak sa ďalšie kroky týkajú prakticky iba triedy App. Preto je na riadku 117 kontrola dátového typu uzla. Nasleduje overovanie kritérií pre triedu App. Kategória a vývojár sa overia rovnako ako názov, pretože sú to stringy.

```
150.          // Size
151.          if (bSize && aNode.Size != Convert.ToInt32(size * 1000))
152.               continue;
153.
154.          if (bSizeH && aNode.Size <= sizeH * 1000)
155.               continue;
156.
157.          if (bSizeL && aNode.Size >= sizeL * 1000)
158.               continue;
159.
160.
161.          result.Add(n);
162.          }
163.
164.       return result;
165.       }
```

Číselné hodnoty sa skontrolujú podobným spôsobom. Konkrétne v našom prípade sa napríklad veľkosť násobí tisíckrát, pretože je vo vstupných dátach uvádzaná v kB a vyhľadávací tag, pracuje s MB, v prípade ceny sa to samozrejme nenásobí. Na záver sa už len vráti výsledný zoznam uzlov a ten spracuje trieda MainPage.

4.4 Výkonnostné testy

Za účelom testovania výkonu sme si pripravili 8 dátových kolekcií s počtami uzlov od 1000 do 20 000. Potom sme si stanovili 8 testov, na ktorých sme merali čas. Keďže zo Silverlightu bola odstránená trieda Stopwatch, ktorá slúžila na meranie času, použili sme jej implementáciu od programátora menom Tiaan, ktorú zverejnil na svojom blogu[13]. Hardwarová konfigurácia, na ktorej prebiehali testy bol Macbook Pro 13 (early 2011), 2.3 GHz dual-core Intel Core i5 with 3MB cache, 4GB of 1333 MHz DDR3 SDRAM, na ktorom bol nainštalovaný Windows 7 Professional. Nainštalovaný Silverlight bol vo verzii 4.0.6 a bol spustený v prehliadači Google Chrome.

Testy boli nasledovné:

1. Load – načítanie xml súboru, merali sa dva časy: načítanie xml v dátovej vrstve a vytvorenie vizuálnej reprezentácie v prezentačnej vrstve
2. Change Year – výsledný čas trval metóde CalculateGraph na prepočítanie počtov uzlov v kolekcii pre jednotlivé dátumy

[13] Tiaanov blog - http://blog.tiaan.com/link/2009/02/03/stopwatch-silverlight

3. History – tento čas potrebovala komponenta na návrat v čase. Čas pri tejto operácií je závislý od počtu uzlov, ktoré treba odobrať z kolekcie, preto záleží na počte uzlov v požadovanom dátume a hlavne na počte uzlov v aktuálnom dátume. Preto sme volili najvyšší čas, aký sa dá dosiahnuť a to je pri plnej kolekcii
4. Search – tento test meral čas, ktorý potrebuje komponenta na vykonanie vyhľadávania nasledovného dotazu: **name: "a"; price<: 7.0; size>: 1.5;** Uvádzame opäť dva časy, prvý potrebovala dátová vrstva na vyhľadanie uzlov, ktoré odpovedajú kritériám a druhý čas potrebovala prezentačná vrstva na zaznačenie týchto uzlov
5. Search (1 year) – rovnaký test ako predchádzajúci s jedinou úpravou a tou je obmedzenie dátumu uzlov na rok 2011 pridaním týchto kritérií: **date-from: 2011-01-01; date-to: 2012-01-01;**
6. Remove – tento test meral čas, ktorý potrebuje prezentačná vrstva na odobratie všetkých uzlov z kolekcie. Opäť je tento čas veľmi závislý na počte odoberaných uzlov, ako aj na počte uzlov v kolekcii. Preto sme merali najvyššiu možnú hodnotu a tou je odoberanie všetkých uzlov z plnej kolekcie
7. Relayout – tento čas potrebovala prezentačná vrstva na znovu rozmiestnenie všetkých uzlov
8. Move – tento posledný test meral, ako dlho trvá posunúť s celou kolekciou

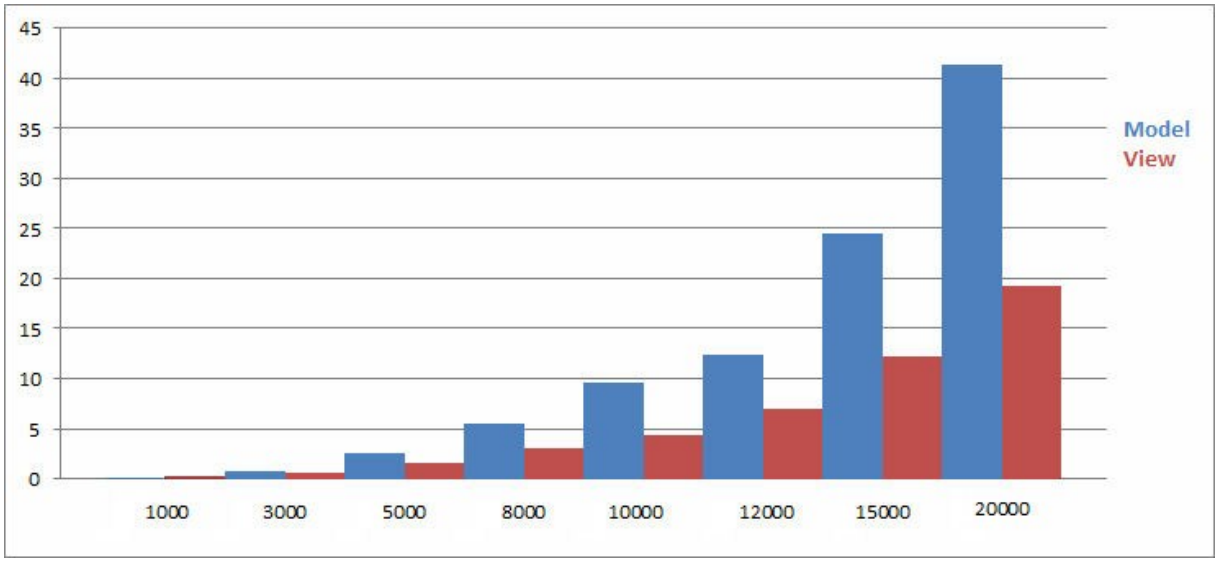
V nasledovných tabuľkách sú namerané hodnoty. Počty uzlov v kolekciách sú uvedené v hornom riadku. Všetky časy sú uvádzané v sekundách a milisekundách, prípadne ešte v minútach:

	1000		**3000**		**5000**		**8000**	
Load	0,1	0,2	0,8	0,6	2,6	1,56	5,48	3,1
Change Year	0,08		0,16		0,22		0,33	
History	00:00,56		00:02,6		00:04,90		00:09,94	
Search	0,013	0,48	0,03	1,5	0,03	4,6	0,26	9,58
Search (1 year)	0,004	0,48	0,01	1,4	0,01	4,3	0,045	9,4
Remove	0,2		1,67		4,8		11	
Relayout	00:00,90		00:07,40		00:21,10		00:54,10	
Move	0,2		2,3		4,1		7,3	

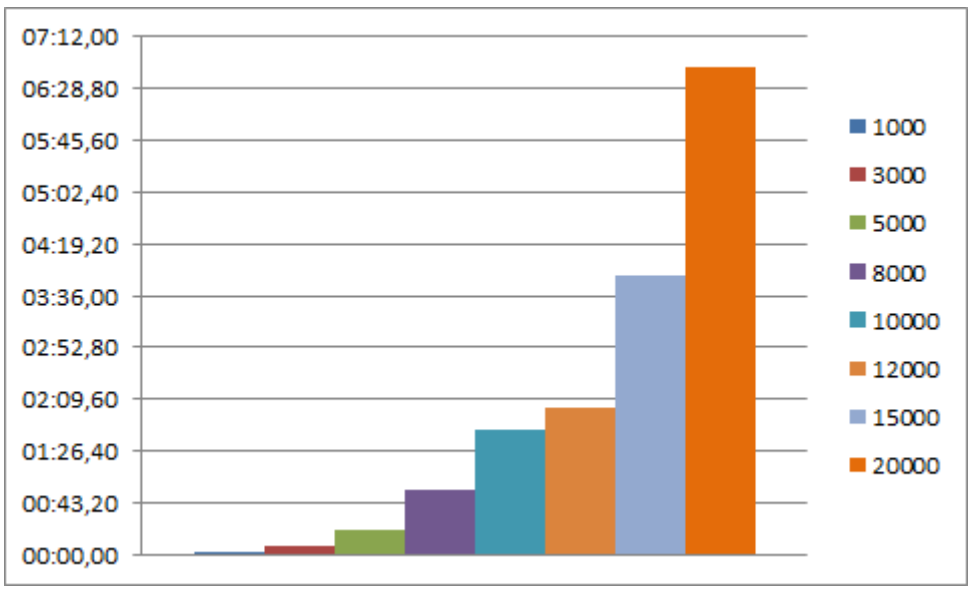
	10000		**12000**		**15000**		**20000**	
Load	9,6	4,37	12,4	7	24,5	12,2	41,3	19,3
Change Year	0,41		0,52		0,63		0,82	
History	00:15,30		00:21,80		00:37,20		01:02,40	
Search	0,06	14,8	0,06	20,3	0,063	34,1	0,071	01:00,80
Search (1 year)	0,04	14,3	0,048	20,1	0,041	32,3	0,064	58,7
Remove	17,8		24,8		39,9		01:11,50	
Relayout	01:43,70		02:01,70		03:53,40		06:46,10	
Move	13,2		17,1		33,6		58,4	

Výkonové testy jasne preukázali, že pohodlne sa dá naša komponenta používať približne do 5000 - 8000 uzlov. Od hranice 8000 už začínajú byť časy veľmi dlhé na to, aby sa dala komponenta normálne používať. Zároveň tieto testy preukázali, že v krajnom prípade je možné túto komponentu použiť na kolekciu až 20 000 uzlov.

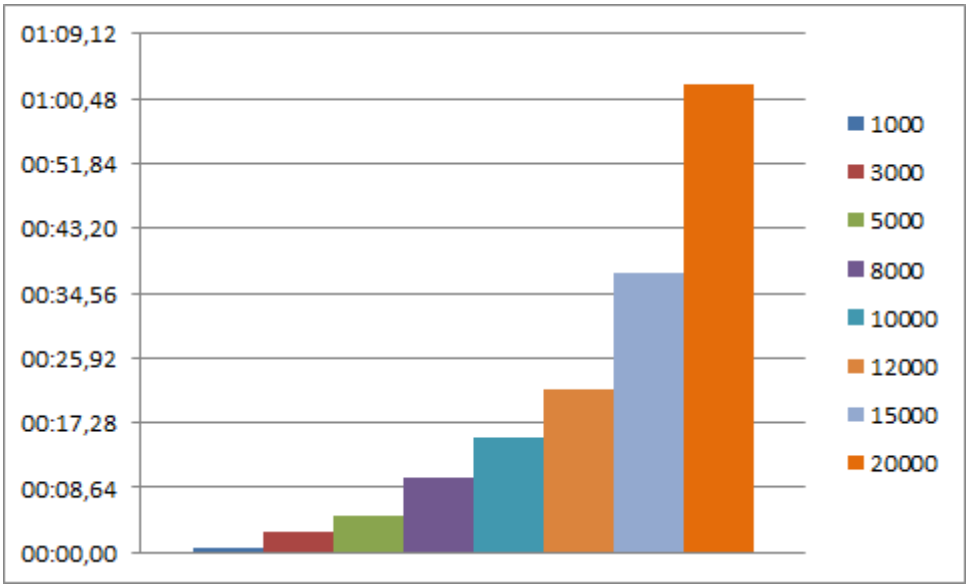
Taktiež bola preukázaná efektivita niektorých algoritmov, napríklad vyhľadávanie podľa kritérií alebo výpočet nárastu kolekcie (Change Year), na ktoré má nárast kolekcie minimálny dopad. O poznanie horšie na tom bol algoritmus na rozmiestnenie uzlov. Nasledovný graf zobrazuje obidva časy načítania a spracovania kolekcie z xml:



Obrázok 9: Časy načítania dát z xml. Modrou farbou je znázornené spracovanie v dátovej vrstve a červenou farbou vygenerovanie vizuálnej reprezentácie uzlov. Zvislá os určuje čas v sekundách, na horizontálnej osy sú kolekcie uzlov.



Obrázok 10: Časy potrebné na znovu rozmiestnenie všetkých uzlov, uvádzané v minútach, sekundách a milisekundách



Obrázok 11: Návrat v čase. Výsledky sú opäť uvádzané v minútach, sekundách a milisekundách

5. Záver

Cieľom tejto bakalárskej práce bolo zbežné zoznámenie sa s hľadaním podobností dokumentov. Bol poskytnutý stručný úvod do problematicky hľadania podobností s referenciami na zdroje, ktoré sa touto problematikou zaoberajú dopodrobna. Taktiež bol poskytnutý prehľad dvoch najvhodnejších technológií na vizualizáciu dát. Hlavným zámerom tejto práce však bol návrh a výsledná implementácia komponenty na vizualizáciu kolekcie dokumentov a ich vzájomných vzťahov – podobností.

Kapitola 3 bola celá venovaná návrhu tejto komponenty a zaoberali sme sa v nej špecifikáciou funkcionality so zameraním na funkcie, ktoré nie sú bežné v tomto type komponent. Ďalej tu bol popísaný návrh dátovej štruktúry a grafického užívateľského rozhrania. Výsledná implementácia bola zdokumentovaná v kapitole 4 spolu s výsledkami výkonových testov.

Výsledkom tejto práce je plne funkčná komponenta na vizualizáciu kolekcií dokumentov, ktorá sa dá bez problémov použiť na akýkoľvek typ kolekcie. Jedným z cieľov tejto práce bolo prepojiť ju s paralelne vznikajúcim projektom na odhaľovanie plagiátov. Toto prepojenie sa však z časových dôvodov nepodarilo uskutočniť. Namiesto toho je použitá náhradná kolekcia dát obsahujúca vývojárov s ich aplikáciami na mobilnú platformu iOS, čím sa potvrdila flexibilita tejto komponenty.

Vo výslednej komponente nie sú ani zďaleka vyčerpané všetky možnosti rozširovania funkcionality, no aj napriek tomu sa jedná o dostatočne vybavenú a plne funkčnú komponentu. Na kolekcii testovaných dát bola overená funkčnosť podľa špecifikácie a komponenta tak splnila očakávania. Na druhú stranu, výkonové testy miestami nedopadli presne podľa očakávaní a boli tak miernym sklamaním. Hlavne prezentačná vrstva by potrebovala v ďalších verziách komponenty zoptimalizovať výkon. Predpokladalo sa, že algoritmus na rozmiestnenie grafu bude časovo náročný, ale aj tak prekonal očakávania a čas skoro 7 minút v prípade dvadsaťtisícovej kolekcie je vysoko nad očakávaniami. Vzhľadom na reálne nasadenie komponenty to však nie je podstatné, pretože tá je navrhnutá na kolekcie o veľkosti maximálne 3000 uzlov a tam dosahuje tento algoritmus čas prijateľných 7 sekúnd. Väčšina testov dosahuje dobré až výborné výsledky. Napríklad algoritmus na vyhľadávanie podľa zadaných kritérií zvládol testovaný výraz vykonať aj na najväčšej kolekcii za 7 stotín sekundy. Taktiež výpočet nárastu kolekcie v konkrétnom roku sa ukázal byť len minimálne ovplyvnený nárastom kolekcie a pri najväčšej kolekcii dosiahol čas 8 desatín sekundy.

Z celkového pohľadu splnila komponenta požiadavky a očakávania. Výkonnostne testy boli veľmi dobré v prípade kolekcií, ktoré sa veľkosťou zmestili do 3000, čo bol požadovaný počet na funkčnosť komponenty. Ostatné požiadavky na funkčnosť spĺňa bez problémov. Prínos tejto komponenty spočíva v tom, že ako flexibilná komponenta obsahuje rozšírenú funkcionalitu na prácu s dátovou vrstvou, ako je vyhľadávanie podľa zadaných kritérií, alebo automatické triedenie do podgrafov. Špecialitou je potom funkcia Progress graph, ktorá umožňuje mapovať vývoj kolekcie a vracať sa v čase.

Ako možné rozšírenia tejto komponenty do budúcich verzií uvádzame: optimalizáciu určitých algoritmov v prezentačnej vrstve, tvorbu krajšieho užívateľského rozhrania, možnosť exportu vykresleného grafu do súboru a našepkávač vo vyhľadávacom riadku.

Literatúra

[1.] BIALAS, Ondřej. *Nástroj pro identifikaci plagiátů a podobných dokumentů*. Ostrava, 2011. Diplomová práce. VŠB – Technická univerzita Ostrava. Vedoucí práce Ing. Radoslav Fasuga, Ph.D.

[2.] STEINBERGER, Ralf, Bruno POULIQUEN a Johan HAGMAN. Cross-Lingual Document Similarity Calculation Using the Multilingual Thesaurus EUROVOC. [online]. 2002, s. 10 [cit. 2012-05-03]. DOI: 10.1007/3-540-45715-1_44. Dostupné z: http://www.springerlink.com/content/d0q93wxbpkaqya85/fulltext.pdf

[3.] GOLDBERGER, Jacob, Shiri GORDON a Hayit GREENSPAN. *An Efficient Image Similarity Measure Based on Approximations of KL-Divergence Between Two Gaussian Mixtures*. Israel, 2003. Dostupné z: http://www.mendeley.com/research/efficient-image-similarity-measure-based-approximations-kldivergence-between-two-gaussian-mixtures-image-modelling-via-mog-3-matching-based-approximation/. Tel-Aviv University.

[4.] LI-CHUN WANG, Avery. An Industrial-Strength Audio Search Algorithm. In: *An Industrial-Strength Audio Search Algorithm* [online]. Palo Alto, 2006 [cit. 2012-05-01]. Dostupné z: http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

[5.] VOSSEN, Gottfried, Darrell LONG a Jeffrey Xu YU. *Web information systems engineering, WISE 2009*: *10th international conference, Poznań, Poland, October 5-7, 2009 : proceedings*. New York: Springer, c2009, 606 s. Lecture notes in computer science, 5802. ISBN 36-420-4408-5.

[6.] OSBORN, Jeremy a Jennifer SMITH. *Web design with HTML and CSS*: *digital classroom*. Indianapolis, IN: Wiley Pub., c2011, 284 s. Digital classroom (Indianapolis, Ind.). ISBN 04-705-8360-6.

[7.] USAMA ALAM, Muhammad. Flash vs. Silverlight: What Suits Your Needs Best?. In: *Smashing magazine* [online]. 2009 [cit. 2012-05-03]. Dostupné z: http://www.smashingmagazine.com/2009/05/09/flash-vs-silverlight-what-suits-your-needs-best/

[8.] CZERNICKI, Bart. *Next-generation business intelligence software with Silverlight 3*. Berkeley , Calif.: Apress, c2009, 419 s. Expert's voice in Silverlight. ISBN 14-302-2487-8.

[9.] KOUBA, Petr. *Technologie Silverlight*. Brno, 2011. Diplomová práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Vedoucí práce Ing. STANISLAV UCHYTIL, Ph. D.

[10.] PAPA, John. *Data-driven services with Silverlight 2*. Sebastopol, CA: O'Reilly, 2009, 342 s. ISBN 05-965-2309-2.

Zoznam príloh

A. Obsah priloženého DVD

Adresár	Obsah adresára
/app/	Zdrojové kódy komponenty a jej implementácia
/sample/	Ukážková kolekcia dát
/text/	Táto bakalárska práca v elektronickej podobe